平成 21 年度　修士論文

無線マルチホップアドホックネットワークにおける TCP-Vegas を拡張した輻輳制御方式

早稲田大学大学院　基幹理工学研究科　情報理工学専攻
5108B004-3
飯窪　尚也

指導　甲藤　二郎　教授

2010 年 2 月 5 日

指導教授印	受付印

目次

第 1 章　序章

1.1.　研究背景

近年、無線通信技術の発展により、無線 LAN、モバイルアドホックネットワーク、無線センサネットワークなど様々な技術が研究されてきている。都市の中にも無線 LAN アクセスポイントや Wi-Fi ホットスポットが設置され、どこでもインターネットを利用することが可能である。また、移動通信端末の小型化、高性能化によりノートパソコン、携帯電話、携帯ゲーム機といった移動体端末を利用した無線通信の利用が拡大し、利用形態も多様化を見せている。

そのような背景の中で、注目を浴びているのがアドホックネットワークである。特にアドホックネットワークでのマルチホップ通信では隠れ端末問題、さらされ端末問題、MAC 層でのコンテンション、衝突、隣接ノードの協調性など考慮しなければならない問題が数多く存在し、現在も多くの研究者が研究を行っているというのが現状である。特に有線向けに設計された従来の TCP 輻輳制御方式では、効率の良いデータ転送できないことが知られている。従来の TCP 輻輳制御方式はウィンドウサイズの増加幅がアグレッシブであり、無線マルチホップドホックネットワーク上ではタイムアウト、MAC 層でのコンテンション、衝突などを引き起こす原因となり、結果的にスループットを低下させてしまう[1]。そのような様々な問題がある中で、マルチホップアドホックネットワークをより普及させていくにはこれらの問題を考慮したマルチホップ通信に適した通信技術、輻輳制御方式の検討、開発をしていかなければならない。

1.2.　研究目的

本研究の目的は、無線アドホックネットワークでのマルチホップ通信を想定した輻輳制御方式に関する検討を行う。従来手法の TCP-Vegas[2]をベースに無線マルチホップアドホックネットワークに適した新たな輻輳制御方式を行う。評価手法として本研究ではネットワークシミュレータの ns-2[3]を用い、従来の輻輳制御方式と提案手法の比較評価を行い、無線アドホックネットワークでのマルチホップ通信でのスループット、パケット到着率等に関する改善を行うことを研究目的とする。

1.3.　本論文の構成

第 1 章である本章では研究背景、研究目的について述べている。
第 2 章では背景技術としてアドホックネットワーク、IEEE802.11 について述べている。
第 3 章では従来手法として TCP 輻輳制御方式について述べている。

第 4 章ではアドホックネットワークでのマルチホップ通信における TCP 輻輳制御方式の問題点、改善手法について述べている。
第 5 章では提案手法について述べている。
第 6 章では ns-2 を用いたシミュレーション実験について述べている。
第 7 章では総括、今後の課題について述べている。

第 2 章　背景技術

2.1.　アドホックネットワーク[4]

アドホックネットワークとは、無線 LAN のようなアクセスポイント、ネットワークインフラを必要としない、無線インターフェースを搭載した端末(PC、PDA、携帯電話など)のみで構成されたネットワークのことである。また「無線アドホックネットワーク」、「自立分散型無線ネットワーク」とも言われている。

アドホックネットワークでは、現在広く PC 等の無線接続に用いられている IEEE 802.11x、Bluetooth などの技術を用いながら多数の端末をアクセスポイントの介在をしないで相互に接続する形態(マルチホップ通信)を取っている。このため、アドホックネットワークでは基地局やアクセスポイントなどのネットワークインフラが不要となり、このようなインフラを持たない場所で安価にネットワークを構築することが可能であり、ある限られた域内での簡易なネットワークの構築の手段として有効である。アドホックネットワークの応用例として、災害現場でのネットワーク、会議場での参加者だけによるネットワークまたは軍事利用での兵士のみによるネットワークなど様々な場所での利用が考えられる。

しかし、現在のところアドホックネットワークの構築には、技術的課題がいくつか残されている。アドホックネットワーク内では常に端末が移動することが考えられ、端末相互間のリンクが確実なものではない。そのような環境でいかに効率よく安定した経路を動的に検出できるか、また、ネットワークの規模と使用する無線の周波数帯域や出力の程度はどう設定すべきか、電力消費をどのようにすべきか、といった考慮しなければならない問題が数多く存在している。

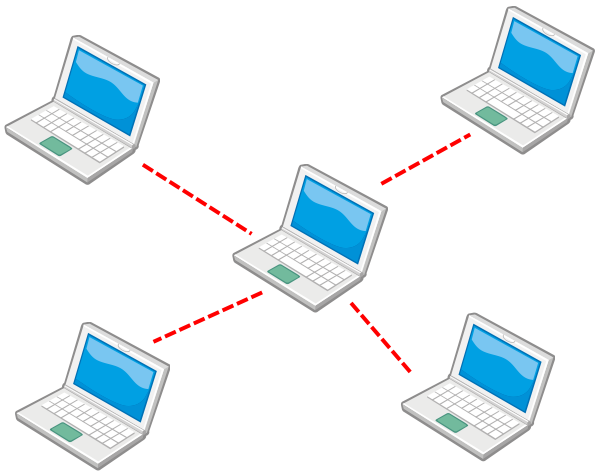

図 1.　アドホックネットワークの構成例

2.2. アドホックルーティング

1970 年代前半に DARPA パケット無線ネットワークが登場して以来、アドホックネットワーク向けの各プロトコルが数多く研究あるいは開発されてきた。アドホックルーティングプロトコルにおいては、各移動ノードがそれぞれ独立に移動、通信を行うためネットワークトポロジが複雑に変化することを想定しなければならない。また、ノード同士の電波干渉や帯域や電力の有効活用、ルーティングループの回避など有線ネットワークでは問題にならなかった点まで考慮する必要がある。

アドホックルーティングプロトコルでは一般に、データ通信する前にあらかじめに経路を作成しておく Proactive 型、送信元ノードから通信要求があった時にのみ経路を作成する Reactive 型、Proactive 型と Reactive 型の両方の性能を兼ね合わせた Hybrid 型と 3 つのパターンに分類することができる。

ここでは Proactive 型の例として DSDV[5]、Reactive 型の例として AODV[6]の説明を行う。

2.2.1. DSDV (Destination Sequenced Distance Vector)[5]

DSDV は、古典的な Bellman-Ford ルーティングアルゴリズムをもとにしたテーブル駆動型のルーティングプロトコルである。この方式では、ネットワーク内のルータ間でルーティング情報がループしないよう改良が加えられている。ネットワーク内の各ノードは、同一ネットワーク内の到達可能な全てのノードの宛先とその宛先までのホップ数を記録したルーティングテーブルを管理しているため、常に利用可能なルーティング情報が準備されている。

シーケンス番号方式は、経路が新しいか古いかをモバイルホストが判断するために使用する。ルーティングテーブルの更新情報は、テーブルの整合性を維持するために定期的にネットワーク全体に送信される。したがって、ネットワーク中に多くの制御トラフィックを生成してしまい、ネットワーク資源の利用効率を低下させてしまうといった問題もある。この問題を軽減するために、DSDV では 2 種類のルーティング情報更新用パケットを使用する。1 つは full dump である。このパケットは、利用可能な全てのルーティング情報を格納しており、ネットワークプロトコルデータユニット (NPDU)を複数要求することができる。ノードの移動が時折発生する程度では、このパケットが送信されることは稀である。もう 1 つは、full dump より小さな差分 (incremental)パケットである。これは、最後に行った full dump 以降の更新情報のみを送信する時に用いる。

新しいルーティング情報をブロードキャストする場合、宛先ノードのアドレス、宛先までのホップ数、そしてそのブロードキャストの新しい一意なシーケンス番号と共に、宛先についての情報を受信した時のシーケンス番号を格納する。そして、最も新しいシーケンス番号を持つ経路が常に使用される。2 つの経路更新情報が同じシーケンス番号を持つ場合は、ホップ数の少ない経路を選択する。

2.2.2. AODV (Ad hoc On-Demand Distance Vector)[6]

AODV は、DSDV アルゴリズムをもとに構築されている。DSDV が完全な経路のリストを維持管理するのに対して、AODV ではオンデマンドに経路を作成する。これにより、必要なブロードキャストの数を最小化するような改良を加えている。

送信元ノードは、ある宛先ノードにメッセージを送信しようとして有効な既存の経路が存在しなかったら、経路探索プロセスを開始して、他の端末の位置を把握する。ノードは、隣接する端末に向けて RREQ (Route Request)パケットをブロードキャストする。隣接ノードはさらに、自信の隣接するノードに RREQ パケットを転送していく。宛先、もしくは宛先への十分に新しい経路を持つ中継ノードが発見されるまで、経路探索は繰り返される。AODV は宛先シーケンス番号を使用することで、全経路がループしないことと、最も新しい経路情報を保持することができる。ブロードキャスト ID は RREQ を送信するたびに増加し、ノードの IP アドレスと組み合わせて RREQ を一意に識別する。自信のシーケンス番号とブロードキャスト ID と共に、送信元ノードは自信の持つ宛先の最新シーケンス番号を RREQ に含ませる。中継ノードは宛先への経路を保持しており、そのシーケンス番号が RREQ に含まれるシーケンス番号と等しいかそれ以上の場合にのみ、RREQ に応答する。

RREQ の転送処理において、中継ノードは最初のブロードキャストパケットを受信した時、それを送信した隣接ノードのアドレスをルーティングテーブルに記録しておき、それにより逆方向の経路を確立する。その後、同じ RREQ パケットを受信すると、そのパケットは破棄される。いったん、RREQ が宛先、または十分に新しい経路を持つ中継ノードに到達したら、その宛先あるいは中継ノードは、最初に RREQ を受信した隣接ノードを経由して RREP (Route Reply)パケットをユニキャストで送り返す。RREP が逆方向経路を伝わって返送される時、経路上のノードはその RREP の送信元ノードへのエントリを自信のルーティングテーブルに設定する。また、各経路エントリともに経路タイマが用意されており、指定したライフタイム内に経路が使用されなかった場合にはそのエントリを削除する。

AODV では送信元ノードが移動すると、宛先への新しい経路を見つけるために経路探索プロセスを再度実行する。経路上のノードが移動すると、その上流の隣接ノードが移動を検知し、アクティブな上流の隣接ノードに対してリンク切断メッセージを送信する。これにより経路のその部分が削除される。今度はそれらのノードが自信の上流隣接ノードにリンク切断通知を送信し、同様のことが送信元ノードに到達するまで続けられる。

AODV のさらなる特徴は、HELLO メッセージを使用することである。これは、各ノードが自分の存在を隣接する他のノードに知らせるために、定期的にブロードキャストするメッセージである。HELLO メッセージはノードのローカルな接続を維持するのに利用できる。ノードは隣接ノードが再送したデータパケットを受信することで、隣接ノードとの接続が生きていることを確認できる。もし受信できない時、ノードは HELLO メッセージの受信を含む何らかの方法を用いて確認を行う。HELLO メッセージは、ノードが受信した他のノードの一覧を含むことができ、それによりネットワーク接続状況についてのより多く

の情報をもたらすことができる。

2.3. IEEE 802.11[7]

IEEE 802.11 無線 LAN (以下、802.11 無線 LAN)は無線 LAN に関する規格であり、物理層における変調方式やデータリンク層のうちの MAC 層の通信制御などについて定められている。802.11 無線 LAN では、同一の無線チャネルを複数端末で共有するためのアクセス制御が実装されている。

802.11 無線 LAN では、アクセス制御機能として CSMA/CA (Carrier Sense Multiple Access with Collision Avoidance)が採用されている。802.11 無線 LAN の基本アクセス手順として DCF (Distributed Coordination Function)では、各局がチャネルの使用状況を検査して自律的にパケットの送信タイミングを決定するが、その時のアクセス制御プロトコルに CSMA/CA を使用している。

このため、たとえ周辺に同一チャネルを利用する無線ノードが存在したとしても、CSMA/CA により、無線ノード間で同一チャネルを共有して通信が可能である。ただし、CSMA/CA では、偶然同時にパケットを送信してしまう可能性、すなわちパケット同士が無線上で衝突してしまう可能性がある。衝突によって、パケットが正常に受信されなかった場合、パケットは再送されるが、衝突の発生する確率は無線局数と通信トラフィックの増加につれて大きくなり、スループットの低下を招いてしまう。

2.3.1. CSMA/CA

CSMA とは、Carrier Sense Multiple Access の略で、キャリア(転送波)感知多重アクセスと訳されているアクセス方式の 1 つで、無線 LAN では広く適用されているアクセス方式である。この CSMA 方式の場合、通信を開始しようとする各無線端末は、開始する前に必ず周りの無線端末が電波を出していないかどうかを、確認してから通信を開始する方式である。

特に CSMA/CA という衝突回避機能を持つ CSMA の場合は、衝突を回避するために、周囲の無線端末が電波を出していれば、ある一定期間 (IFS：Inter Frame Space)だけ待って、再び周囲が電波を出していなければあるランダムな時間後に自分が電波を送信する方式です。これは、現在普及している無線 LAN には、標準として広く適用されている方式です。

CSMA/CA の特徴は、全ての無線端末が対等の関係にあり、しかも分散制御している点にある。また、CSMA/CA 方式は、各端末が使用している時間に関係して通信が可能か、どうかが決定されるので、広義の TDMA 方式に該当する。

次に、CSMA/CA の基本的なアクセス手順について説明する。図 2 において、端末 B、C は、端末 A が送信中(①)にはキャリアセンスを行い、無線チャネルをビジーと判断するため、端末 A が送信を終了して無線チャネルが未使用になるまで待機する。端末 B、C は、未使用状態への移動と同時に IFS 時間待ち、更にランダムで決まるバックオフ時間だけキャリ

アセンスを行い、バックオフ時間の短かった端末 B が未使用状態であることを確認して送信(②)を行っている。同様の手順で、端末 C が送信(③)を行っている。

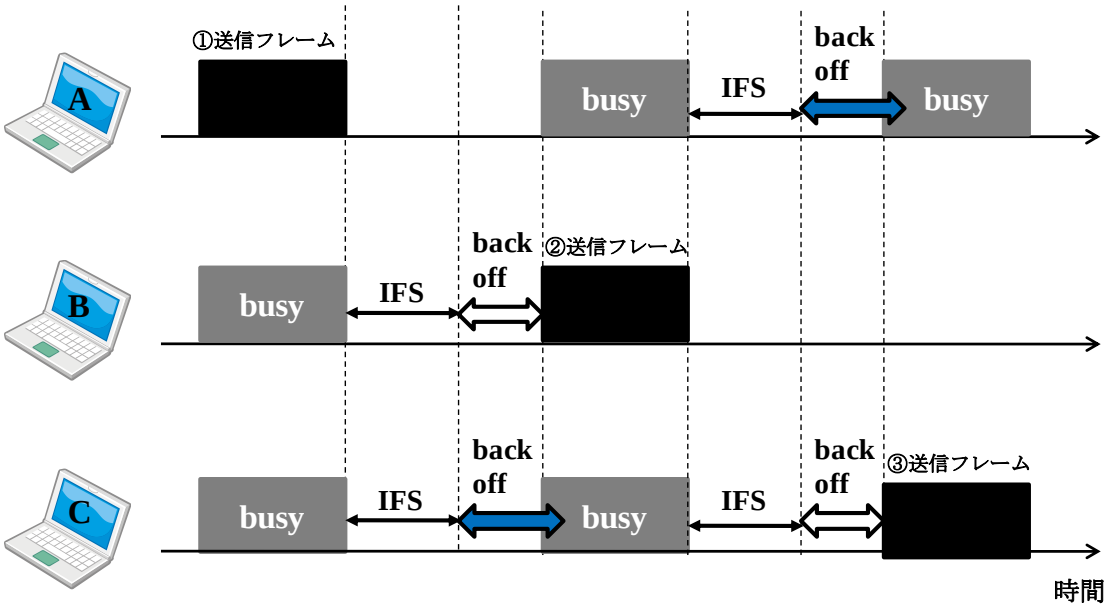

図 2.　CSMA/CA における基本的なアクセス手順

2.3.2.　隠れ端末問題

無線通信では、無線端末間の距離や電波を通さない障害物などの影響により、互いの無線信号が到達しない状態(キャリアセンスが機能しない伝搬環境)が起こりえます。これは、隠れ端末問題（Hidden Terminal Problem)と呼ばれており、図 3 に例を示してある。図 3 のようにノード A、B、C があった場合、ノード A はノード B のみと、ノード B はノード C のみしかキャリアセンス可能な範囲にあらず、ノード A とノード C はお互いにキャリアセンスできない範囲にあるとする。この状態で、ノード A からノード B へデータを送信している間に、ノード A の送信をキャリアセンスできないノード C からノード B へデータを送信してしまうと、パケットの衝突が起きるという問題である。このため、CSMA/CA ではパケット衝突の頻度が増加し、スループット低下の原因となる。

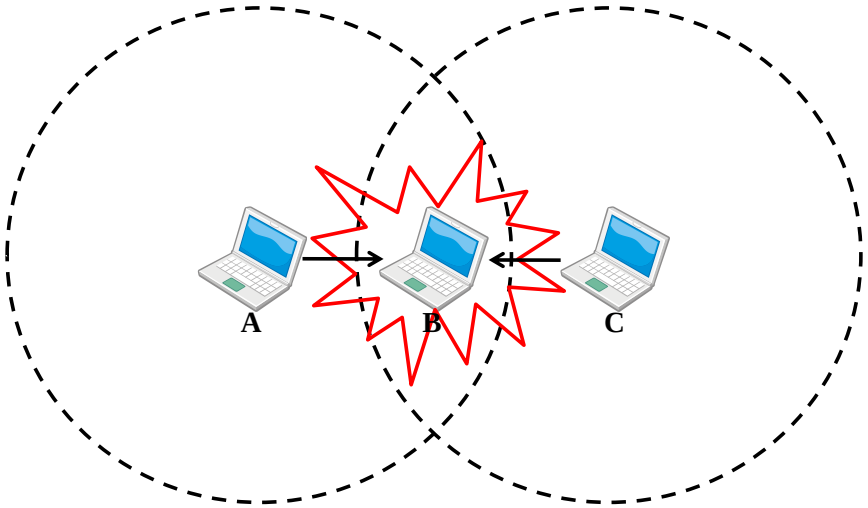

図 3.　隠れ端末問題

2.3.3.　RTS/CTS

そこで 802.11 規格ではキャリアセンスが有効に機能しない伝搬環境に対応するために RTS (Request to Send)、CTS (Clear to Send)と呼ばれる対策機能が定義されている。その動作手順を図 4 として示す。

端末 A は、データ・フレーム送信前に DIFS に加えてバックオフ時間をキャリアセンスして、RTS をデータ・フレームの宛先の AP(アクセスポイント)に送信する(①)。端末 A と端末 B は互いにキャリアセンスできる環境下にあるため、端末 A の RTS を端末 B も受信することができる(①')。RTS と CTS のフレームには無線回線を使用する予定期間が記載されているデュレーション・フィールドがあり、端末 B は RTS フレームに記載されている期間だけ送信を禁止（NAV：Network Allocation Vector)することにより衝突を防止できる。

一方、データ・フレームの宛先である AP は、RTS 受信後 SIFS 時間空けて端末 A 宛に CTS を返す(②)。AP が送信した CTS は、端末 C も受信することができるので(②')、端末 C は CTS フレームに記載されている期間だけ NAV をセットすることにより衝突を防止できる。

CTS を受信した端末 A は、SIFS 時間後にデータ・フレームを送信する(③')。ここで、SIFS は DISF より小さいため、短い時間の SIFS を使って優先権を持たせることで、いったん RTS が正常に受信されると以後の手順中のフレームは妨害されることなく交換される。データ・フレーム受信完了後、AP は ACK フレームを返して通信を終了する(④)。

データ・フレームの送信元端末 A、受信先 AP 以外の周辺無線局(端末 B、C)は RTS あるいは CTS の受信後、デュレーション・フィールドに記載されている期間は無線回線が使用されていると見なし、NAV をセットすることにより衝突を防止できる。

また、RTS/CTS を用いることによって、送信端末の信号をキャリアセンスできない環境の端末が存在しても、基地局が送信する CTS を受信できれば送信端末の存在を知ることが

できるため、隠れ端末問題を解決する手段ともなる。

隠れ端末のケースでは、フレームサイズが大きいほど衝突する確率が高くなるため、送信するデータ・フレーム長が、パラメータ RTSThreshold（RTS 閾値)の大きさを超える場合に、RTS/CTS を使用する。

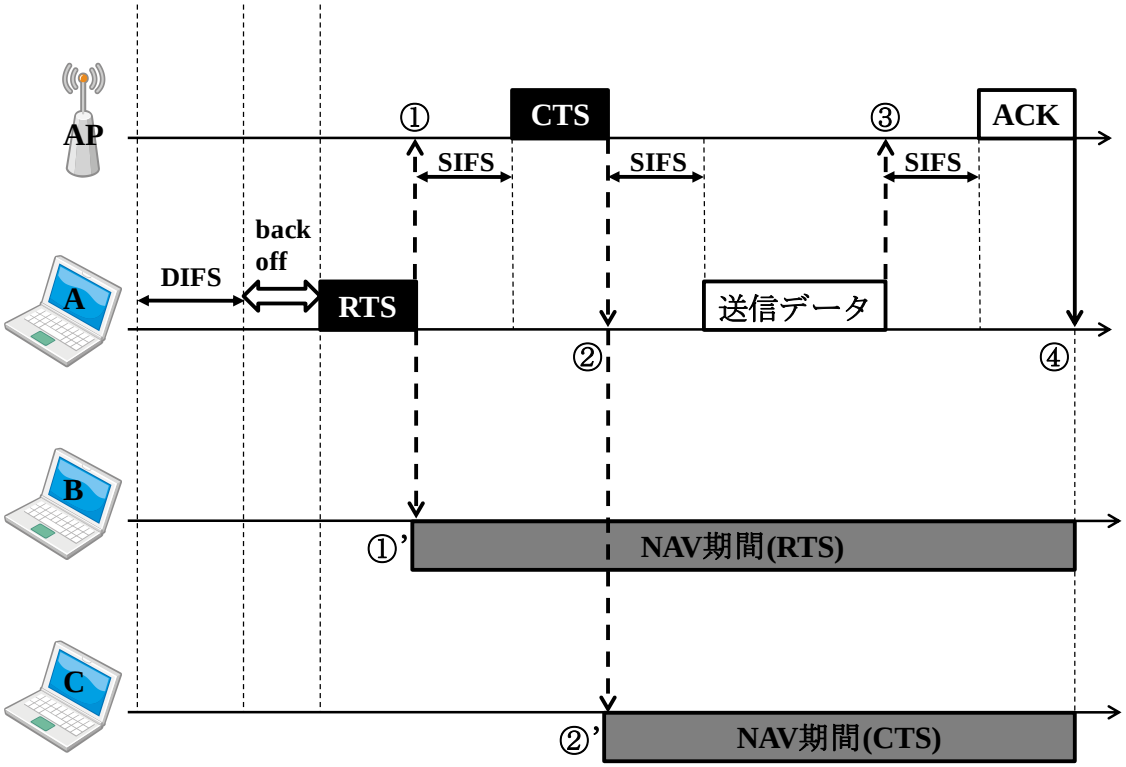

図 4.　RTS/CTS を用いた CSMA/CA の通信手順

2.4.　アドホックネットワークにおける帯域のキャパシティ

アドホックネットワークにおけるマルチホップ通信では、ホップ数が増えるたびにスループットが減少していくことが知られている。[8]ではマルチホップアドホックネットワークでのモデル化を行い、ノード数によるキャパシティ(利用可能な最大スループット)の推定を行っている。

ノードの密度が一定であるという前提で、ノード数の合計を n とする。1 ホップのキャパシティを $O(n)$と与えられる。しかし、ホップ数の増加に伴い、ネットワークが大きくなる。経路の平均の長さは、ネットワークの空間的な直径、あるいは同等の面積の平方根か、つまり

$$O(\sqrt{n}) \tag{2.1}$$

に伴い、大きくなることが期待されている。

このことから、エンド間でのキャパシティはおおよそ

$$O(n/\sqrt{n}) \tag{2.2}$$

と表すことができ、それぞれのノードにおけるエンド間での利用可能なスループットは以下のように表すことができる。

$$O(1/\sqrt{n}) \tag{2.3}$$

[8]では、一列に並んだ Chain トポロジにおけるマルチホップで、パケットサイズやホップ数を変えた場合でのスループットの変化に関する実験を行っている。その結果を図 5 に示す。この実験での帯域幅は 802.11 の 2[Mbps]である。

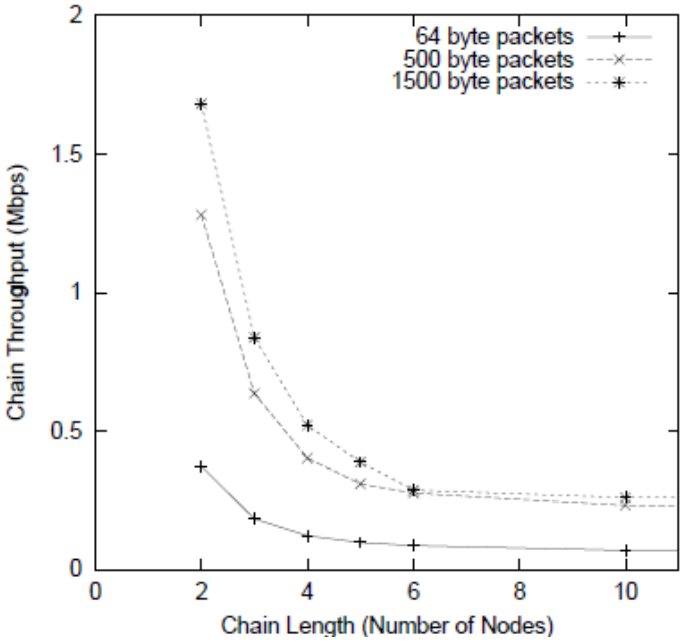

図 5.　マルチホップ通信におけるスループット

第 3 章　TCP による輻輳制御方式

3.1.　TCP による輻輳制御方式

TCP による輻輳制御方式では、ネットワークに滞在しているパケット数を示すウィンドウサイズをもとにフロー制御を行っている。パケットが損失しないで送信できている時は、ネットワークが空いていると判断し、転送速度を上げ、逆にパケットが損失する場合が多い場合、ネットワークが混んでいると判断して、転送速度を下げる動作を行う。送信者は、このウィンドウサイズを増減させることにより送信量を調整し、輻輳制御を行っている。

そこで、TCP による輻輳制御方式として代表的な、TCP-Tahoe[9]、TCP-Reno[10]、TCP-Vegas[2]、TCP-Vegas を拡張した Ada-Vegas[11]、TCP-Vegas を無線マルチホップアドホックネットワーク向けに拡張した Vegas-W[12]の説明を行う。

3.1.1.　TCP-Tahoe[9]

TCP-Tahoe は 1988 年に発表された輻輳制御方式であり、現在実装されている方式のベースとなっている。Tahoe では、スロースタートフェーズと輻輳回避フェーズと 2 つのフェーズから構成されており、それぞれ増加方法が異なる。

スロースタートフェーズは、新しい通信を開始する場合やタイムアウトの後に再び通信を開始する場合に利用される。スロースタートフェーズでは、ウィンドウサイズがスロースタート閾値（*ssthresh*)と呼ばれる変数より小さい場合に利用される。スロースタートフェーズで、*ssthresh* を超えた場合、輻輳回避フェーズへと移行する。

スロースタートフェーズの開始時点では、ウィンドウサイズを 1 に設定する。確認応答セグメント（ACK)を受信するたびに 1MSS (Maximum Segment Size)分の大きさだけウィンドウサイズを増加させていく。増加前のウィンドウサイズを *cwnd*、増加後のウィンドウサイズを $cwnd_{new}$、最大セグメント長を *MSS* とすると、スロースタートフェーズでの操作は、以下の式(3.1)のように表すことができる。

$$cwnd_{new} = cwnd + MSS \tag{3.1}$$

図 6 にスロースタートフェーズでの通信の様子を示す。

図 6 の RTT1 の期間で、TCP はスロースタートフェーズを開始し、ウィンドウサイズを 1 に設定する。それにより、データセグメントを 1 つ送信する。RTT2 の期間では、TCP は 1 つの ACK を受信しているため、ウィンドウサイズを 1 つ増加させ、2 に設定する。こ

れにより、この期間に 2 つのデータセグメントを送信する。RTT3 の期間では、2 つの ACK を受信したため、ウィンドウサイズを 2 つ増加させ、4 に設定し、データセグメントを 4 つ送信する。

このように、スロースタートフェーズでは、ウィンドウサイズを 1 ラウンドトリップごとに 2 倍ずつ更新されていく指数関数的な増加となる。

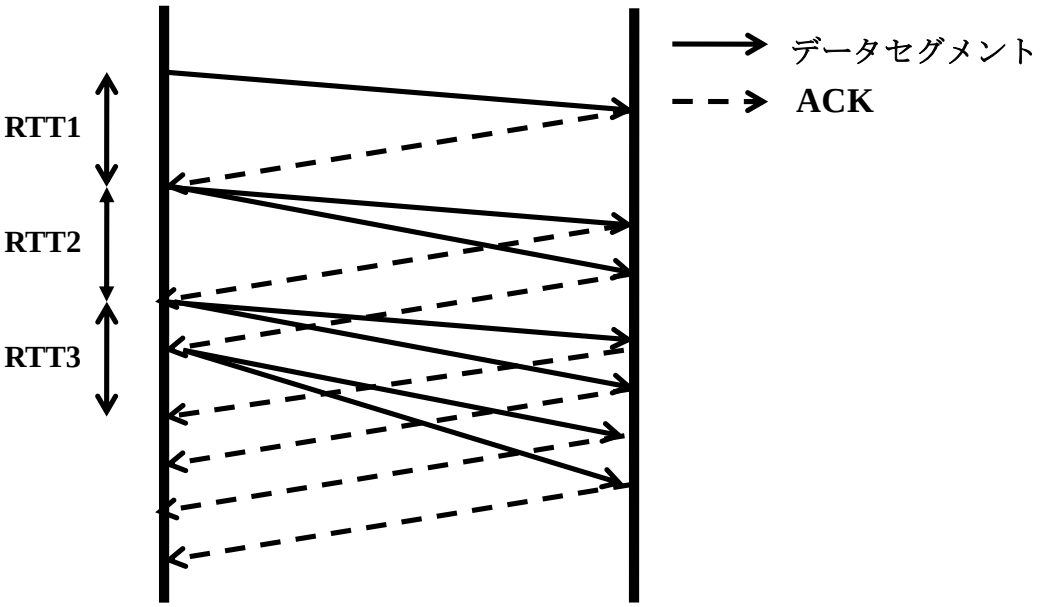

図 6. スロースタートフェーズ

輻輳回避は、ウィンドウサイズが *ssthresh* を超えた場合に、スロースタートフェーズから輻輳回避フェーズへと移行される。

輻輳回避フェーズでは、ACK を受信するごとに、1/ウィンドウサイズの大きさ分だけウィンドウサイズを増加させる。増加前のウィンドウサイズを *cwnd*、増加後のウィンドウサイズを $cwnd_{new}$、最大セグメント長を *MSS* とすると、輻輳回避フェーズでの操作は、以下の式(3.2)のように表すことができる。

$$cwnd_{new} = cwnd + MSS/cwnd \qquad (3.2)$$

図 7 に輻輳回避フェーズでの通信の様子を示す。

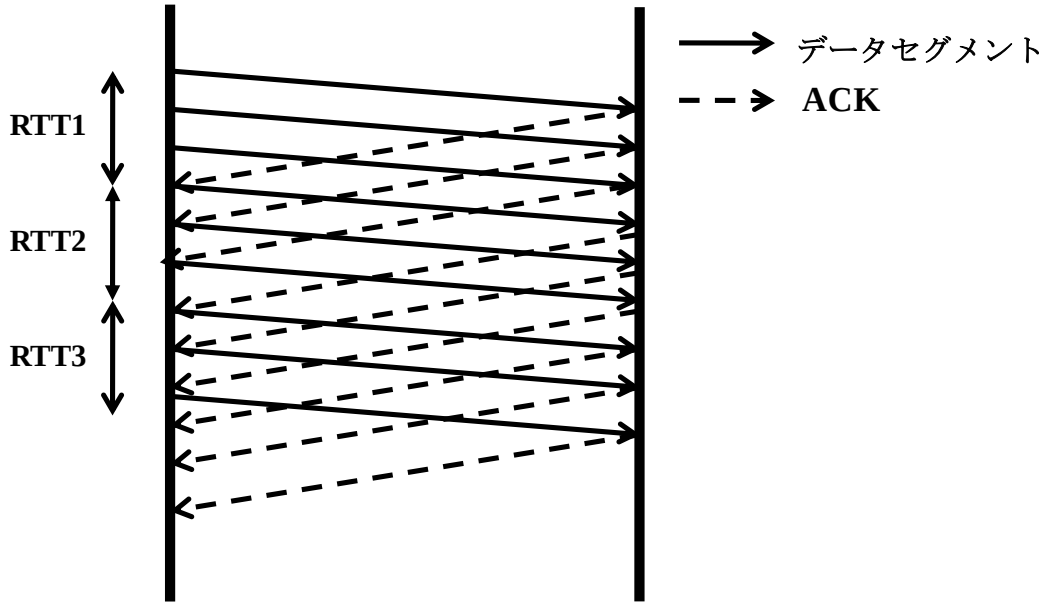

図 7.　輻輳回避フェーズ

図 7 の RTT1 の期間では、ウィンドウサイズが 3 に設定されている。RTT2 の期間では、ウィンドウサイズは ACK を受信するたびに現在のウィンドウサイズ分の 1 だけ増加させ、RTT2 ではウィンドウサイズは 4 に設定される。

図 8 では Tahoe によるウィンドウサイズの変化の様子を示す。

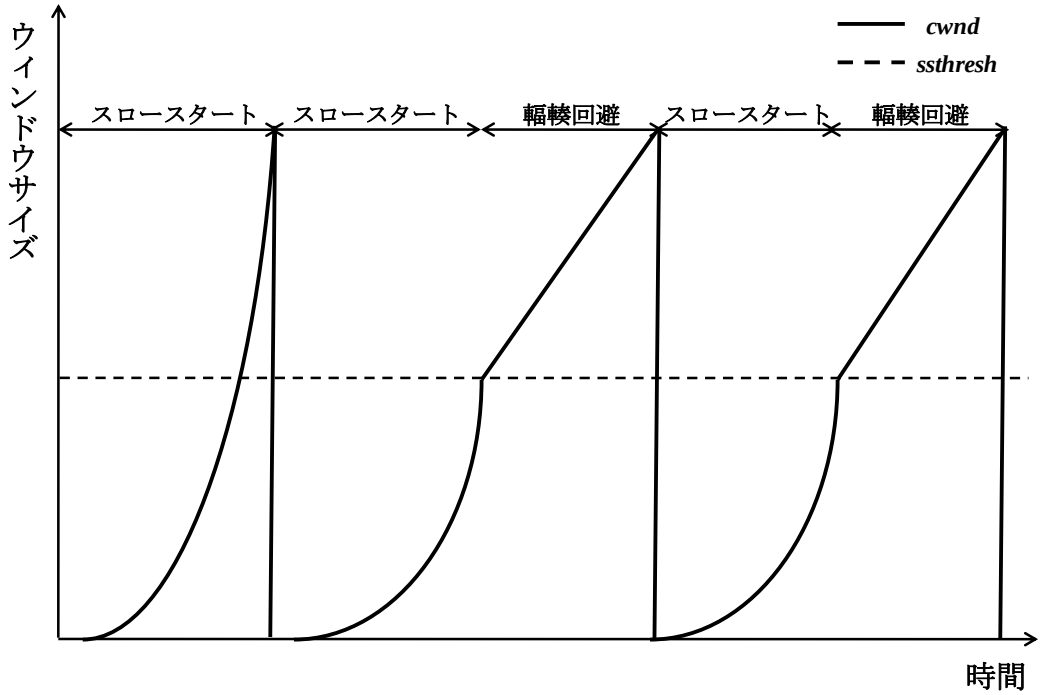

図 8.　Tahoe によるウィンドウサイズの変化

図 8 では、スロースタートフェーズからデータ転送を開始する。このフェーズでの Taho は、ウィンドウサイズを 1 から RTT ごとに 2 倍ずつ増加させていき、ネットワーク容量を超える大きさに達し、パケット廃棄が起こっている。そして、*ssthresh* をパケット廃棄のウィンドウサイズの半分の値に設定する。その後、スロースタートフェーズからデータ転送を再開する。パケット損失検出後は、*ssthresh* が設定されているため、ウィンドウサイズが *ssthresh* に達した時点で、輻輳回避フェーズへ移行する。

このように Tahoe は、2 つの通信フェーズを繰り返しながら周期的な通信を行う。

3.1.2. TCP-Reno[10]

Tahoe では輻輳を検出すると、ウィンドウサイズを 1 まで減少させ、そこから再度スロースタートフェーズを再開させ、転送速度を向上させていく。Tahoe では輻輳に対して保守的過ぎる面があり、転送速度の性能からは問題がある。Tahoe を用いた場合、1%のパケットロス率に対して、50~70%の転送性能の劣化が生じると言われている。Reno は、このような問題を解決するために、Tahoe の輻輳制御方式に改良を加え、高速リカバリアルゴリズムを導入した。高速リカバリアルゴリズムにより、パケット廃棄に対して、効率的なデータ転送を行うことができる。

図 9 に Reno のウィンドウサイズの変化の様子を示す。

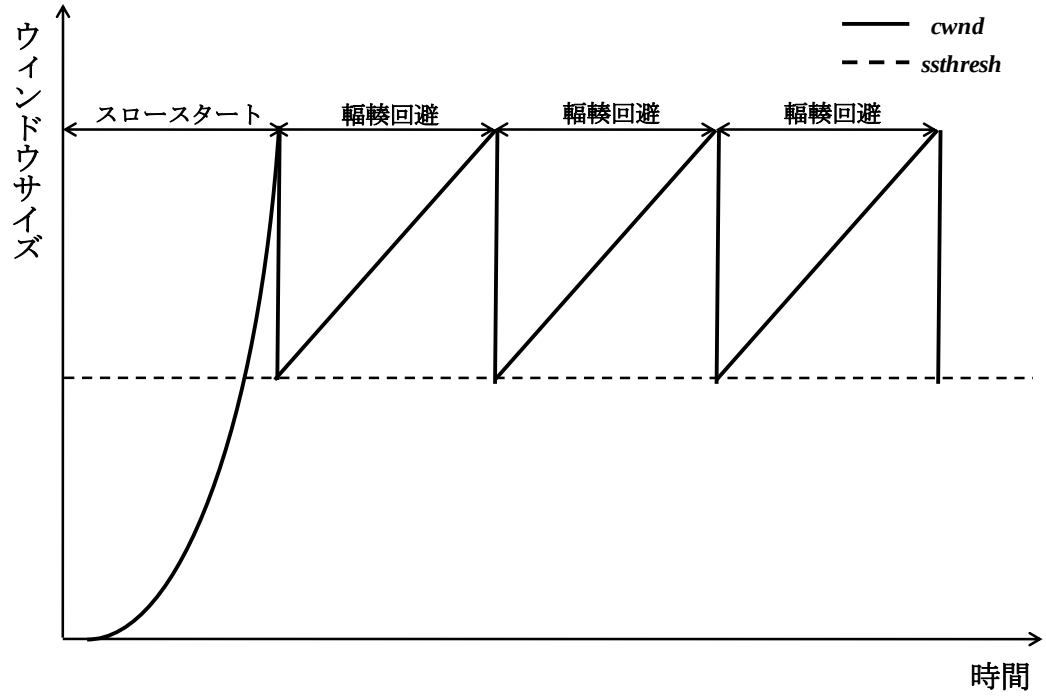

図 9. Reno によるウィンドウサイズの変化

Reno の輻輳制御方式は、データ送信の開始時には、ウィンドウサイズの大きさを 1 に設

定する。そこから、スロースタートフェーズによりウィンドウサイズを増加させていく。*ssthresh* は、TCP の送信バッファの大きさなど、この TCP コネクションで許容でき最大値に設定される。

Reno のパケット廃棄の検出方法は Tahoe と同様で、タイムアウトと重複 ACK を用いて行う。Reno では、高速再送アルゴリズムによる再送を行った後の挙動が Tahoe と異なる。パケット廃棄を検出すると、Tahoe では *ssthresh* を現在のウィンドウサイズの大きさを半分に設定し、ウィンドウサイズを 1 に設定する。Reno では、*ssthresh* とウィンドウサイズを現在のウィンドウサイズの半分に設定する。したがって、高速再送アルゴリズムによりセグメントの再送を行った後に、輻輳回避フェーズへと移行する。このアルゴリズムを高速リカバリアルゴリズムと呼ぶ。

高速リカバリアルゴリズムを用いることにより、Reno はパケット廃棄時の転送速度の減少が Tahoe よりも小さくなる。このため、パケット廃棄による転送性能の劣化が小さくなり、Tahoe よりも高い転送効率でデータ転送を行うことができる。

3.1.3. TCP-Vegas[2]

Tahoe、Reno では、パケット廃棄を検出して大きくなりすぎたウィンドウサイズの調整を行う。したがって、パケット廃棄の発生直後にウィンドウサイズが必要以上に小さくなるため、転送性能が劣化する。Vegas では、送信したパケットのラウンドトリップタイム(RTT)を観測し、その変動をウィンドウサイズに利用する。RTT が大きくなれば、輻輳が発生していると判断してウィンドウサイズを小さくし、RTT が小さければ、逆にウィンドウサイズを増加させる。

Vegas では、ルータのキュー内のパケット数(Δ)の推定値によって制御される。Δ を以下の式(3.3)で示す。ただし、*cwnd* は現在のウィンドウサイズ、*RTT*、RTT_{min} はそれぞれ現在の *RTT*、*RTT* の最小値を示す。

$$\Delta = \frac{cwnd}{RTT_{\min}} - \frac{cwnd}{RTT} \tag{3.3}$$

増加前のウィンドウサイズを *cwnd*、増加後のウィンドウサイズを $cwnd_{new}$、最大セグメント長を *MSS* とすると、Vegas でのスロースタートフェーズでの操作は、以下の式(3.4)のように表すことができる。ここでの、p、γ は定数とする。

$$cwnd_{new} = \begin{cases} cwnd + MSS & (\Delta < \gamma) \\ cwnd \times (1 - p) & (\Delta \geq \gamma) \end{cases} \tag{3.4}$$

また、輻輳回避フェーズでの操作は以下の式(3.5)のように表すことができる。増加前のウィンドウサイズを *cwnd*、増加後のウィンドウサイズを $cwnd_{new}$、最大セグメント長を *MSS* とする。ここでの、α、βは定数とする。

$$cwnd_{new} = \begin{cases} cwnd + MSS/cwnd & (\Delta < \alpha) \\ cwnd & (\alpha \le \Delta \le \beta) \\ cwnd - MSS/cwnd & (\Delta > \beta) \end{cases} \quad (3.5)$$

図 10 に Vegas のウィンドウサイズの変化の様子を示す。

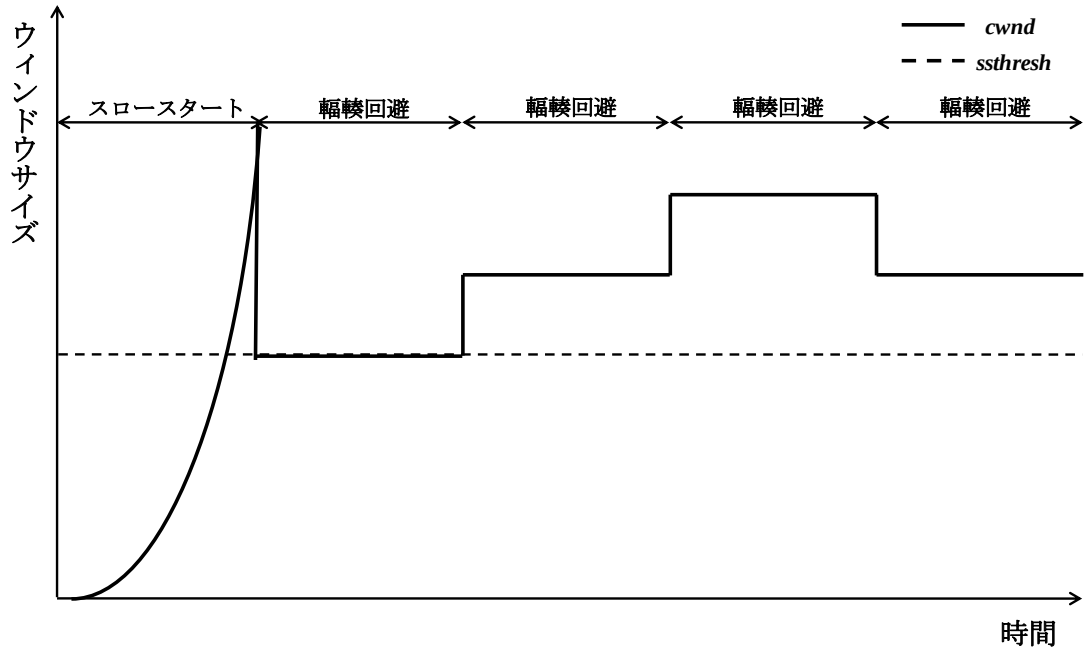

図 10.　Vegas によるウィンドウサイズの変化

スロースタートフェーズでは、Tahoe、Reno と同様に指数関数的に増加させる。ただし($\Delta \geqq \gamma$)フェーズではウィンドウサイズを(1-p)倍して、ウィンドウサイズを減少させる。そして、輻輳回避フェーズでは、Δ の値がある下限値(α)よりも小さい場合には、回線帯域は利用されていないと判断して、輻輳ウィンドウサイズを Tahoe、Reno 同様の $1/RTT$ 増加する。そして、上限値(β)を超えた場合には、初期輻輳と判断し、ウィンドウサイズを $1/RTT$ ずつ減少させる。以上の 2 つのケース以外、つまり Δ が α と β の間にある場合には、輻輳状態ではなく、かつ回線帯域は利用されていると判断し、ウィンドウサイズを変化させない。

以上より、Vegas では、他のトラヒックが存在しない場合にウィンドウサイズはパケット

廃棄を引き起こすことなく、一定となり、Reno より 30%から 70%のスループットの改善をすることが可能であると言われている。

しかし、Reno と Vegas が混在している場合は、Vegas 自身のスループットが減少するという問題もある。これは、Reno のウィンドウサイズの増加を、初期輻輳と判断してしまうために、Vegas のウィンドウサイズは減少傾向となり、スループットが減少してしまうからである。

3.1.4. Ada Vegas[11]

Ada Vegas は Vegas をベースに、アダプティブに輻輳制御を行う輻輳制御方式である。Ada Vegas はネットワーク環境の変化をみて、ウィンドウサイズの増加幅を増減させ、ネットワーク環境が変化しても、Vegas より効率の良いデータ転送ができる。

スロースタートフェーズでの操作は Vegas と同様である。

また、Ada Vegas での輻輳回避フェーズでの操作は以下の式(3.6)のように表すことができる。増加前のウィンドウサイズを $cwnd$、増加後のウィンドウサイズを $cwnd_{new}$、最大セグメント長を MSS とする。ここでの、α、β は定数とする。

$$cwnd_{new} = \begin{cases} cwnd + inc \times MSS/cwnd & (\Delta < \alpha) \\ cwnd & (\alpha \le \Delta \le \beta) \\ cwnd - MSS/cwnd & (\Delta > \beta) \end{cases} \qquad (3.6)$$

Ada Vegas では連続して($\Delta \leqq \alpha$)のフェーズに入った回数を $succ$ とする。$succ$ の値により、ウィンドウサイズの増加幅のパラメータ inc を動的に変化させる。また、$succ$ の値により、inc 以外にも定数 α、β の値も変化させる。パラメータ $succ$ と inc、α、β の関係を表 1 に示す。

表 1. パラメータ *succ*、*inc*、α、β の関係

	α	β	inc
$0 \leqq succ \leqq 3$	1	3	1
$4 \leqq succ \leqq 7$	2	4	2
$8 \leqq succ \leqq 15$	2	4	4
$16 \leqq succ$	4	8	8

従来の Vegas を Vegas_1、輻輳回避フェーズの($\Delta \leqq \alpha$)フェーズでの上げ幅を従来の Vegas の 8 倍にしたものを Vegas_8 とする。ボトルネックリンクを 0.1[Mbps]、8[Mpbs]とした場合の Ada Vegas、Vegas_1、Vegas_8 のウィンドウサイズの変化の様子を図 11、12 に示す。

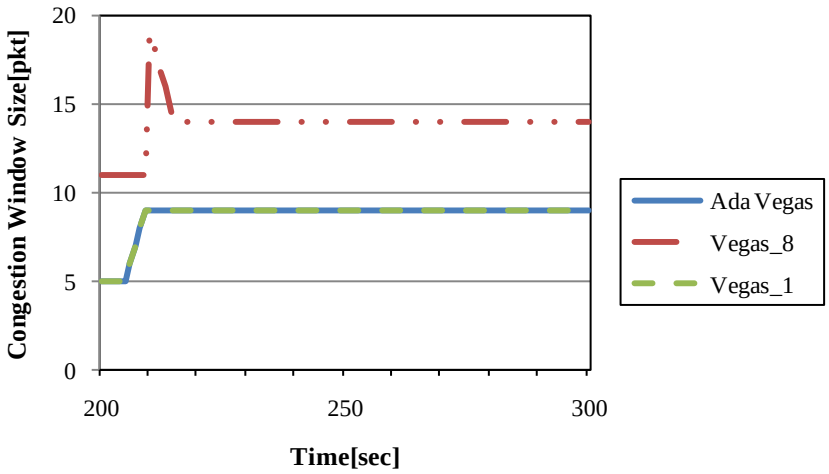

図 11.　Ada Vegas,Vegas_8,Vegas_1 によるウィンドウサイズの変化(0.1Mb)

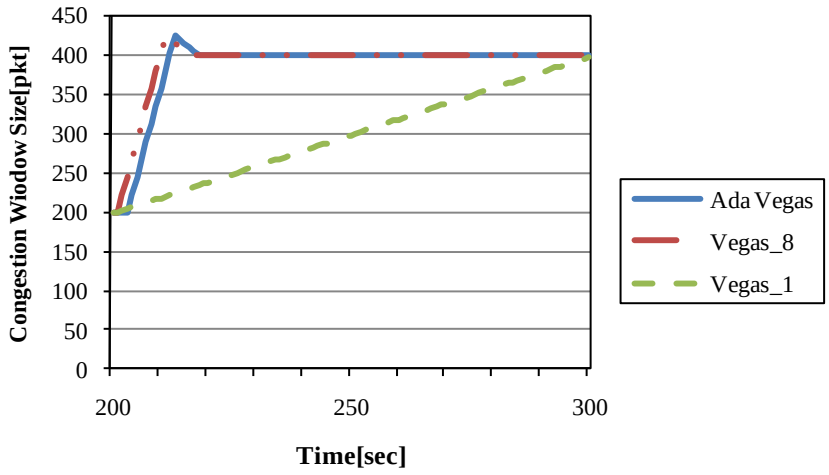

図 12.　Ada Vegas,Vegas_8,Vegas_1 によるウィンドウサイズの変化(8Mb)

図 11 から低帯域の場合、Ada Vegas は Vegas_1 のようにパケット廃棄を起こさず、滑らかにウィンドウサイズを増加させ、一定の値に収束している。しかし、Vegas_8 は、215 秒付近でオーバーフローを起こし、パケット廃棄が起きている。このため、ウィンドウサイズを一度減少させ、一定の値に収束している。

図 12 から高帯域の場合、Ada Vegas は Vegas_8 のように、パケットサイズを起こさず、高速にウィンドウサイズを増加させている。しかし、Vegas_1 は Vegas_8 に比べ、ウィンドウサイズの増加幅が小さいため、ウィンドウサイズを増加させるのに時間がかかっている。そのため、転送効率は劣化し、スループットの低下につながる。

Ada Vegas は、輻輳回避フェーズでのウィンドウサイズの増加幅をアダプティブに変化させることにより、低帯域、高帯域で効率の良いデータ転送を行うことができる。図 11、12 から低帯域の場合、オーバーフローよるパケット廃棄を起こさず、一定の値に収束する

ことができ、高帯域の場合は、高速にウィンドウサイズを増加させることで、データ転送の効率を高くすることが可能である。

3.1.5. Veagas-W[12]

[12]では、無線マルチホップアドホックネットワーク向けに Vegas を拡張した輻輳制御方式 Vegas-W を提案している。Vegas-W ではウィンドウサイズの過度な増加を抑える手法を取り入れて、MAC 層でのコンテンション等のパケットロスを軽減することでスループットの改善を目的としている。Vegas-W では以下に挙げる 4 つの面で改良を行っていた。

1. Fractional Window サポート
2. スロースタート
3. 輻輳回避
4. スロースタート閾値更新

まず、無線マルチホップアドホックネットワークにおいて、あるホップ数での最適なウィンドウサイズを W'とする。マルチホップアドホックネットワークにおいて、W'は小数であることが知られている。また、W'が 1 より大きい場合もあるが、W'が 1 より小さい場合と比べると、スループットに与える影響は少ない。W'が 1 より小さい場合、RTT ごとに整数でウィンドウサイズを増加させていくことは、オーバーフローを起こす原因となってしまう。そこで、Vegas-W ではウィンドウサイズの最小値を 1 に設定している。Fractional Window サポートではパラメータとして、連続して再送タイムアウトが起こった回数を N_{fw}とする。Fractional Window サポートでは、連続して再送タイムアウトが起こった回数が N_{fw}に達したら、ウィンドウサイズを 0.5 にする。Vegas-W での最小のウィンドウサイズは 1 であるため、0.5 に設定することでパケットの送信を一度見合わせることになる。そして、そこからパケットのリスケジューリングを行って、そのタイマーでパケットの送信を行う工夫を行っている。

Vegas-W のスロースタートフェーズでは図 13 に示すアルゴリズムで制御を行っている。ここではウィンドウサイズを W、N_sをウィンドウサイズ増加閾値の定数とし、n_Sを連続して($\Delta \leqq \gamma$ & $n_s \leqq N_s$)フェーズに入った回数とする。Δ は式(3.3)と同様で、γ を定数とする。

$$W = \begin{cases} W & , \text{if } \Delta < \gamma \ \& \ n_s <= N_s \\ W+1 & , \text{if } \Delta < \gamma \ \& \ n_s > N_s \\ W \times (1-p) & , \text{if } \Delta >= \gamma \end{cases}$$

図 13. Vegas-W のスロースタート制御方式

($\Delta \leqq \gamma$ & $n_s \leqq N_s$)フェーズに入ったら n_Sを 1 増加させ、($\Delta \leqq \gamma$ & $n_s > N_s$)フェーズになって

初めてウィンドウサイズを 1 増加させる。ここで、nsを初期値の 0 に設定する。これにより、Vegas-W ではウィンドウサイズを Vegas より増加させない工夫を取り入れている。この手法により、ウィンドウサイズの過度な増加によるオーバーフローを軽減することが可能である。

輻輳回避フェーズでもスロースタート同様な手法でウィンドウサイズを増加させている。Vegas-W の輻輳回避フェーズの制御を図 14 に示す。ここでは N_{CA}をウィンドウサイズ増加閾値の定数とし、n_{CA}を連続して($\alpha<\Delta<\beta$ あるいは $\Delta\leqq\alpha$ & $n_{CA}\leqq N_{CA}$)フェーズに入った回数とする。α、βを定数とする。

$$W = \begin{cases} W + \frac{1}{W} & , \text{if } \Delta <= \alpha \ \& \ n_{CA} > N_{CA} \\ W & , \text{if } \alpha < \Delta < \beta \\ & \quad or\ \Delta <= \alpha \ \& \ n_{CA} <= N_{CA} \\ W - \frac{1}{W} & , \text{if } \Delta >= \beta \end{cases}$$

図 14.　Vegas-W の輻輳回避制御方式

($\alpha<\Delta<\beta$ あるいは $\Delta\leqq\alpha$ & $n_{CA}\leqq N_{CA}$)フェーズに入ったら n_{CA}を 1 増加させ、ウィンドウサイズはそのままの大きさでキープする。($\Delta\leqq\alpha$ & $n_{CA}>N_{CA}$)フェーズでウィンドウサイズを増加させる。ここで、n_{CA}を初期値の 0 に設定する。輻輳回避フェーズでもウィンドウサイズを閾値に達するまで増加しないようにしている。ウィンドウサイズをあまり増加させず、ウィンドウサイズを減少させる時は迅速に行い、Wにウィンドウサイズを近づけウィンドウサイズを安定させることを目的としている。しかし、パケットの送信者が Wの値を正確に把握することは難しいので、ウィンドウサイズを小さな値に留めてオーバーフロー等のパケットロスを軽減することを目的とした工夫がなされている。

スロースタート閾値更新としては、スロースタート閾値(W_{th})をアダプティブに変化させる改良を加えている。Vegas ではパケットがロスした場合、W_{th}を半分に設定する。これは、有線向けに設定されているため、マルチホップアドホックネットワークでは適していない。そこで、Vegas-W では W_{th}を 2 つのフェーズで更新を行う。1 つは新しい ACK が到着した場合である。もう 1 つは再送タイムアウトが起こった時である。新しい ACK が到着した場合のスロースタート閾値更新のアルゴリズムを図 15、再送タイムアウトが起こった場合のスロースタート閾値の更新を図 16 に示す。ここでは輻輳回避フェーズでのウィンドウサイズを W_S、($\alpha<\Delta<\beta$)に入った回数を ns_{CA}、閾値を NS_{CA}、再送タイムアウトを起こした回数を to_{SS}、閾値を TO_{SS}とする。

Algorithm 1: W_{th} **Update When Receiving a New ACK**
Initialization: Receiving a new ACK with sequence number equal to or larger than beg_seqno
Method:

1. Calculate $Expected$ and $Actual$
2. Calculate the integral number of packets congested: $\Delta = (int)((Expected - Actual) * baseRTT + 0.5)$
3. **IF** $w < W_{th}$
4. **IF** $\Delta > \gamma$
5. $W_{th} \leftarrow$ current window minus 1
6. **END**
7. **ELSE**
8. **IF** $\Delta < \alpha$
9. ns_{CA} keeps unchanged
10. **END**
11. **IF** $\Delta > \beta$
12. $ns_{CA} \leftarrow 0$
13. **END**
14. **IF** $\alpha \leq \Delta \leq \beta$
15. **IF** $Currentwindow > W_s$
16. $ns_{CA} \leftarrow 0$
17. $W_s \leftarrow$ current window
18. **END**
19. $ns_{CA} \leftarrow ns_{CA} + 1$
20. **IF** $ns_{CA} > NS_{CA}$
21. $W_{th} \leftarrow w$
22. $ns_{CA} \leftarrow 0$
23. **END**
24. **END**
25. **END**
26. $beg\_seqno \leftarrow$ sequence number of the ACK

図 15.　新しい ACK が到着した場合の W_{th} 更新アルゴリズム

Algorithm 2: W_{th} **Update When Timeout Occurs**
Initialization: Timeout Occurs
Method:

1. **IF** $w < W_{th}$
2. $to_{ss} \leftarrow to_{ss} + 1$
3. **ELSE**
4. $to_{ss} \leftarrow 0$
5. **END**
6. **IF** $to_{ss} \geq TO_{ss}$
7. $W_{th} \leftarrow W_{th} - 1$
8. $to_{ss} \leftarrow 0$
9. **IF** $W_{th} < 1$
10. $W_{th} \leftarrow 1$
11. **END**
12. **END**

図 16.　再送タイムアウトが起こった場合の W_{th} 更新アルゴリズム

図 15、16 のアルゴリズムで W_{th} をアダプティブに変化させ、オーバーフローを起こさせないような手法であり、マルチホップアドノックネットワークに適した W_{th} に設定しようとしている。

第 4 章　無線マルチホップアドホックネットワークにおける TCP 輻輳制御方式

4.1.　無線マルチホップアドホックネットワークにおける TCP の問題点

信頼性を有するデータ転送を提供するため、現在は TCP がトランスポートプロトコルとして使用されている。しかし、本来、TCP は有線ネットワークでの利用を前提としているため、無線マルチホップアドホックネットワークに適用した場合、転送性能が劣化することが知られている[13]。

マルチホップアドホックネットワークでは、有線ネットワークと異なり、すべてのノードが通信媒体である無線空間を共有している。そのため、一度にデータ転送を行えるのは 1 ユーザのみとなる。そのため、マルチホップ通信ではホップ数の増加、ウィンドウサイズの増加に伴なり、送信者あるいは送信パケット数が増えるほど、利用競争が発生する。それにより、転送性能が下がり、スループットが劣化してしまう問題がある[14]。ウィンドウサイズをアグレッシブに増加させることにより、MAC 層でのコンテンション、パケット衝突を引き起こし、結果再送タイムアウトに導き、スループットの低下につながってしまう。また、TCP はパケットロスを観測するまで、転送レートを増大し続けるため(Tahoe、Reno など)、伝送遅延の増大と帯域の長時間占有という 2 つの問題が発生し、スループットと公平性が劣化する。

また、MAC 層の制御方式に IEEE 802.11 を用いた場合、ランダムバックオフタイマーという乱数を用いて、送信間隔を空けるため送信権に不公平性を生じる。ノードが広範囲にわって存在する場合には、隠れ端末問題、さらされ端末問題が顕著になって、MAC 層でのパケット廃棄が増える。

また、有線ネットワークのパケット廃棄の主な原因は、輻輳によるオーバーフローによるものであるが、アドホックネットワークでは輻輳によるオーバーフローは稀なケースである。アドホックネットワークでは、通信エラー、リンクの切断、端末の移動、宛先までの経路がないなど、様々な要因でパケット廃棄が起こる[15]。

上記のように、様々な要因が関係して、無線マルチホップアドホックネットワークでの TCP の性能は劣化してしまう。マルチホップアドホックネットワークでの TCP の性能を改善するには、TCP だけでなく、MAC 層のアクセス手順、ルーティングプロトコルなど様々な問題を考慮しなければならない。

4.2. 無線マルチホップアドホックネットワークにおける性能改善手法

現在では、無線マルチホップアドホックネットワークにおけるスループットの性能改善手法として数多くの研究がなされてきた。改善手法の例として、トランスポート層、MAC 層の 1 つのレイヤのみ改善を行い、スループットを上げる手法とトランスポート層と MAC 層のクロスレイヤ設計を行い、スループットの改善を行うなど様々な手法が提案されている。

[16]、[17]では TCP のウィンドウサイズの上限を制限することで、スループットの改善を行う手法を提案している。[16]では、ウィンドウサイズの上限とスループットの関係を解析していた。また、[17]では、ホップ数とウィンドウサイズの上限によるスループットの関係を解析していた。[16]、[17]ともに、ウィンドウサイズの上限を制限することで、パケットの送信数を抑えることができ、MAC 層でのコンテンション、衝突を軽減することができる。その結果、スループットの増加を確認している。

[16]では、ノード間隔を 200[m]で Chain トポロジ、ウィンドウサイズを 1[pkt]から 8[pkt]に上限を制限した場合の Reno のスループットを測定していた。帯域幅は 2[Mbps]、ルーティングプロトコルに AODV、パケットサイズを 1024[Byte]とする。ウィンドウサイズの上限を制限した場合の Reno のスループットを図 17 に示す。

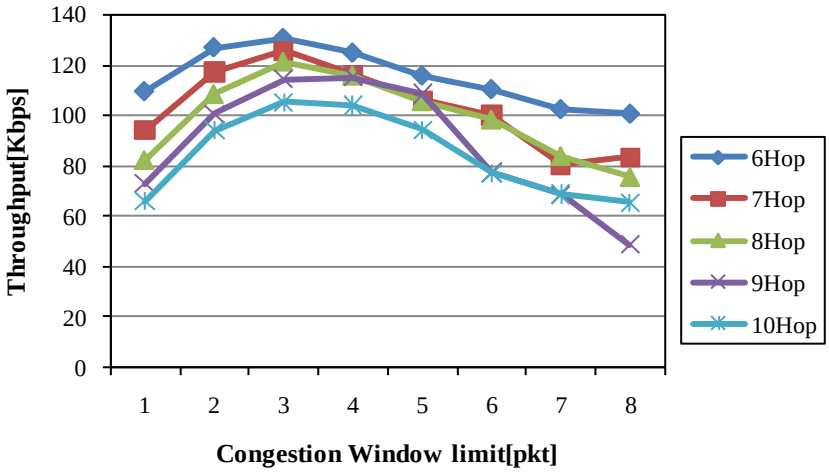

図 17. ウィンドウサイズの上限とスループットの関係

図 17 から、どのホップにおいてもウィンドウサイズの上限を 3[pkt]に制限した方がスループットが良いことが分かる。ウィンドウサイズの上限を制限しない場合は、ACK を受信するたびに、ウィンドウサイズをアグレッシブに増加させてしまい、MAC 層でのコンテンション、衝突が増加してしまう。ウィンドウサイズの上限を制限した場合は、ウィンドウサイズの増加を抑えることがで、MAC 層でのコンテンション、ロスを制限しない場合に比べて、軽減できている。その結果、スループットの改善につながる。

また、[17]では宛先までのホップ数を N とし、TCP のウィンドウサイズの上限を CWL (Congestion Window Limit)として、Nによって CWL を動的に変える手法を提案していた。Nと CWL の関係を表 2 に示す。

表 2.　Nと CWL の関係

	CWL
$N \leqq 4$	2
$N \leqq 8$	1
$N \leqq 12$	2
$N \leqq 20$	3
$N \leqq 26$	4
$N \leqq 30$	5

[18]では、従来の TCP(Tahoe、Reno)における輻輳回避フェーズにおける上げ幅を改良し、IEEE 802.11 のマルチホップ通信に適した方式、FeW (Fractional Window Increment)を提案していた。増加前のウィンドウサイズを $cwnd$、増加後のウィンドウサイズを $cwnd_{new}$ とすると、FeW での輻輳回避フェーズの制御を式(4.1)で表せる。[18]では、定数 α を変化させ、Chain トポロジ、Grid トポロジなどで評価実験を行っていた。[18]では従来手法(α=1)より、マルチホップ通信では α がより小さい α=0.01 などの方が、より効率の良い通信が行えるとしていた。

$$cwnd_{new} = cwnd + \alpha / cwnd \tag{4.1}$$

また、[12]の実験では Vegas-W と FeW のスループット比較を行っており、図 18 にその結果を示す。パラメータの NSは 10、N_{CA} 、NS_{CA}はともに 100、TO_{SS}は 2、N_{fw}は 2 とする。帯域は 2[Mbps]で、フロー数を 1、2、4、8 フローでの実験を行っていた。

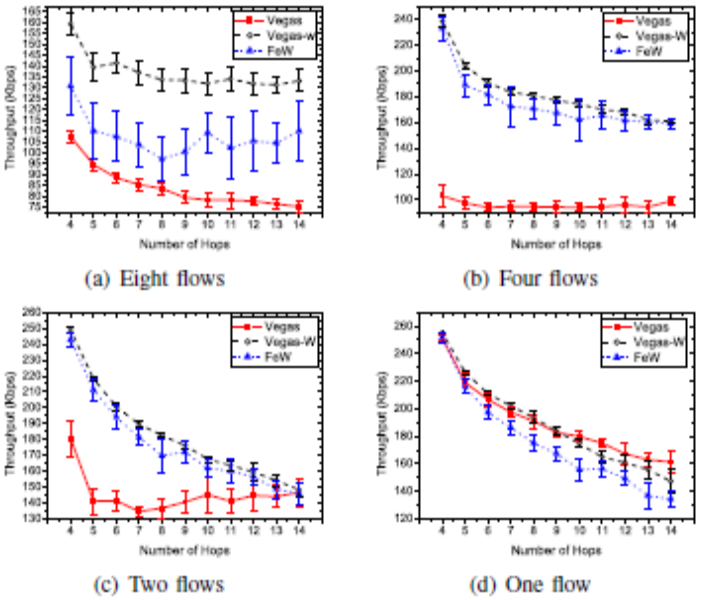

図 18.　Vegas、Vegas-W、FeW のスループット比較

図 18 の結果より、フロー数が多くなるにつれて Vegas-W は Vegas、FeW よりスループットの改善が大きくなっている。フロー数が多くなるにつれて、ネットワーク負荷が増加した場合に Vegas-W は転送効率の良い通信ができている。しかし、1 フローの場合は、従来手法の Vegas よりスループットが低くなっているので、ネットワーク負荷が小さい場合には、効率の良い通信ができていない。

上記のようにマルチホップ通信における TCP の改善手法として、共通して重要なのはウィンドウサイズの上げ幅であることが分かる。マルチホップ通信では、TCP のようにアグレッシブにウィンドウサイズを増加させることで、スループットの低下を導いてしまうことが分かる。そのため、ウィンドウサイズの増加幅を小さくすることが重要である。そうすることにより、MAC 層でのロスも軽減でき、スループットの増加につながるのである。

また、マルチホップ通信での改善手法として、トランスポート層以外にも、[19]では MAC 層のチャネル使用率を観測して、送信タイミングを制御する手法や、[20]ではパケットの送信スケジュールを改善することにより、マルチホップ通信の転送効率を上げる手法を提案していた。[20]では、送信スケジュールを図 19 のようにすれば、Chain トポロジでは干渉、衝突などを起こさずに送信できるとしている。

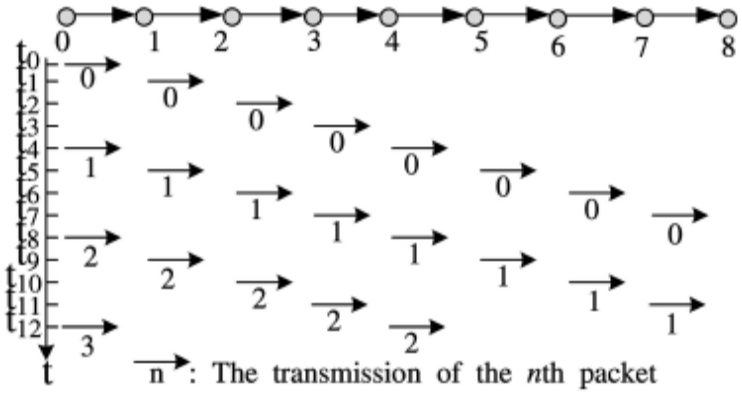

図 19.　Chain トポロジにおける理想的なパケットスケジューリング

このように無線マルチホップアドホックネットワークにおける通信効率の改善手法として TCP 層だけでなく、MAC 層、パケットスケジューリングなど様々な面から改善が可能であると同時に、通信効率の改善をするためには 1 つのレイヤだけではなく、複数のレイヤを考慮しなければならないことが分かる。

第 5 章　提案手法

5.1.　提案手法

本章では、無線マルチホップアドホックネットワークにおける転送効率の改善手法の提案手法について説明していく。

[21]では、無線マルチホップアドホックネットワークにおいては Tahoe、Reno のスループットより RTT を指標に輻輳制御を行う Vegas のスループットが高いことが示されている。Vegas の方が、無線マルチホップアドホックネットワークでは Reno よりスループットが 30%~70%高いとされている。そこで、本稿では TCP-Vegas をベースに無線アドホックマルチホップネットワークに適した輻輳制御方式を提案する。

前章より、マルチホップ通信における問題点と改善手法を示した。そこから、改善手法のポイントとしてウィンドウサイズの増加幅が注目される。そこで、提案手法ではスロースタートフェーズ、輻輳回避フェーズの 2 つのフェーズでのウィンドウサイズの増加幅を改善する。

また、前章の図 17 より、ウィンドウサイズをある値で制限した方がスループットの改善をできることから、ウィンドウサイズをある値に制限するような制御を加えた。

上記のように、提案手法ではウィンドウサイズの増加幅、ウィンドウサイズの制御について改良を行った。詳細を以下の節から説明していく。

5.1.1.　スロースタートフェーズ

無線マルチホップアドホックネットワークにおいては、ウィンドウサイズを 1 増加させることは、アグレッシブでありスループットの低下を導いてしまう。提案手法のスロースタートフェーズでは、Ada Vegas、Vegas-W のように($\Delta < \gamma$)フェーズに連続して入った回数を指標にウィンドウサイズを増加させるよう制御した。

増加前のウィンドウサイズを $cwnd$、増加後のウィンドウサイズを $cwnd_{new}$ とする。また、連続して($\Delta < \gamma$)フェーズに入った回数を $succ$ とする。$succ$ の初期値は 0 とする。増加幅のパラメータ $count$ を設定する。$count$ の初期値は 1 とする。ここでの、p、γ は定数とする。また、Δ は 3 章の式(3.3)とする。

提案手法でのスロースタートフェーズのウィンドウサイズの操作を式(5.1)に示す。

$$cwnd_{new} = \begin{cases} cwnd & (\Delta < \gamma \,\&\, succ \le 10) \\ cwnd + 1/(2 \times count) & (\Delta < \gamma \,\&\, succ > 10) \\ cwnd \times (1 - p) & (\Delta \ge \gamma) \end{cases} \quad (5.1)$$

(*Δ*＜*γ* & *succ*≦10)のフェーズを①、(*Δ*＜*γ* & *succ*＞10)のフェーズを②、(*Δ*≧*γ*)のフェーズを③とする。

①では、*succ* を 1 増加させ、ウィンドウサイズはそのままの大きさでキープする。

②に入るのは、①で *succ* を 10 より大きくした後、かつ(*Δ*＜*γ*)でこのフェーズに入る。②で初めてウィンドウサイズを増加させる。ウィンドウサイズを 1/(2×*count*)分だけウィンドウサイズを増加させ、*succ* を初期値の 0 に戻し、*count* を 1 増加させる。最初にこのフェーズに入った場合は、ウィンドウサイズを 1/2(*count* の初期値が 1 より)だけ増加させ、次にこのフェーズに入るときはウィンドウサイズの上げ幅が 1/4(1 回目で *count* の値が 2 に設定されているため)になる。

③では、ウィンドウサイズを(1-*p*)倍し、ウィンドウサイズを下げ、輻輳回避フェーズへと移行する。その際に、*succ*、*count* の値を初期値に設定する。

succ、*count* の値は再送タイムアウト、スロースタートフェーズから輻輳回避フェーズへと移行した際に初期値に設定する。

5.1.2. 輻輳回避フェーズ

輻輳回避フェーズにおいても、スロースタート同様にウィンドウサイズの増加幅を Vegas より小さくする制御に改良した。ウィンドウサイズの増加の仕方も基本的にはスロースタートフェーズと同様な手法で増加させている。ウィンドウサイズの減少幅は Vegas と同様とする。

増加前のウィンドウサイズを *cwnd*、増加後のウィンドウサイズを $cwnd_{new}$ とする。また、連続して(*Δ*＜*α*)のフェーズに入った回数を *succ2* とする。*succ2* の初期値は 0 とする。増加幅のパラメータ *count2* を設定する。*count2* の初期値は 1 とする。ここでの、*α*、*β* は定数とする。

提案手法での輻輳回避フェーズのウィンドウサイズの操作を式(5.2)に示す。

$$cwnd_{new} = \begin{cases} cwnd & (\Delta < \alpha \,\&\, succ2 \le 10) \\ cwnd + \dfrac{1}{cwnd \times 2 \times count2} & (\Delta < \alpha \,\&\, succ2 > 10) \\ cwnd & (\alpha \le \Delta \le \beta) \\ cwnd - \dfrac{1}{cwnd} & (\Delta > \beta) \end{cases} \quad (5.2)$$

($\Delta < \alpha$& $succ2 \leqq 10$)のフェーズを①'、($\Delta < \alpha$& $succ2 > 10$)のフェーズを②'、($\alpha \leqq \Delta \leqq \beta$)のフェーズを③'、($\Delta > \beta$)のフェーズを④'とする。

①'では、①と同様な制御で、*succ2* を 1 増加させ、ウィンドウサイズはそのままの大きさでキープする。

②'では、②と同様な制御で、①'で *succ2* を 10 より大きくした後、かつ($\Delta < \alpha$)でこのフェーズに入る。輻輳回避フェーズでは、②'で初めてウィンドウサイズを増加させる。ウィンドウサイズを $1/(cwnd \times 2 \times count2)$分だけウィンドウサイズを増加させ、*succ2* を初期値の 0 に戻し、*count2* を 1 増加させる。

③'では、ウィンドウサイズをそのままの大きさでキープし、*succ2* を初期値の 0 に設定する。

④'では、ウィンドウサイズを $1/cwnd$ 分だけ下げ、ウィンドウサイズを下げる。その際に、*succ2* 初期値の 0 に設定する。

またスロースタート同様に、*succ2*、*count2* の値は再送タイムアウト時に初期値に設定する。

輻輳回避フェーズでの提案手法のウィンドウサイズの様子を図 20 に示す。*succ2* が 8、*count2* が 1 の状態のウィンドウサイズの様子を表す。また、図の T1 から T5 までの区間における *succ2*、*count2* のパラメータを表 3 に示す。

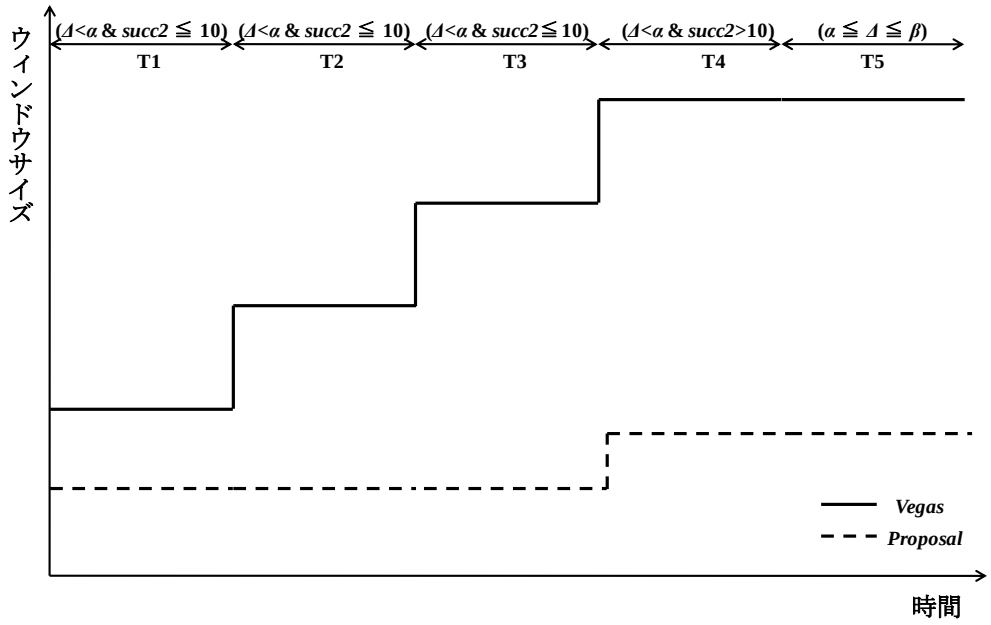

図 20. 輻輳回避フェーズでの提案手法によるウィンドウサイズの変化

表 3.　提案手法の *succ2*、*count2* の変化

	succ2	*count2*
T1	**9**	**1**
T2	**10**	**1**
T3	**11**	**1**
T4	**0**	**2**
T5	**0**	**2**

T1 では、($\Delta < \alpha$)フェーズのため、Vegas はウィンドウサイズを増加する。しかし、提案手法では、(*succ2*≦10)のため、ウィンドウサイズを増加させず、*succ2* を増加する。

同様に、T2、T3 でも同様な操作を行う。T3 の終了時点で、*succ2* が 11 に設定される。

T4 では、($\Delta < \alpha$ & *succ2*>10)フェーズのため、ここで初めて提案手法は、ウィンドウサイズを増加させる。その後、*succ2* を 0 に設定し、*count2* を 1 増加させる。

T5 では、($\alpha \leqq \Delta \leqq \beta$)フェーズのため、Vegas、提案手法共に、ウィンドウサイズをそのままの大きさでキープする。このフェーズでは提案手法は、*succ2* を 0 に設定する。

5.1.3.　ウィンドウサイズ制御手法

無線マルチホップアドホックネットワークでは、ウィンドウサイズが大きくなりすぎてしまい、再送タイムアウトを引き起こす現象が見られる。

ここで、1 フロー6 ホップでの Chain トポロジでの Reno のウィンドウサイズの様子を図 21 に示す。帯域は 2[Mbps]とする。

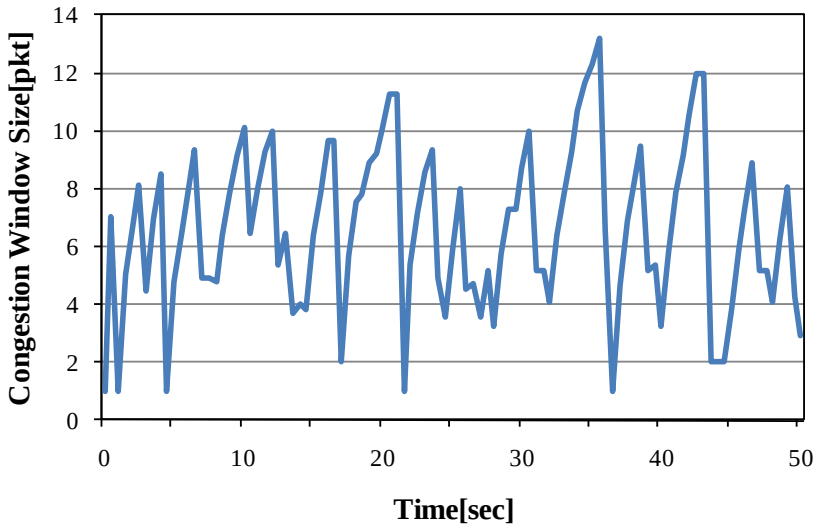

図 21.　1 フローChain トポロジ 6 ホップでの Reno のウィンドウサイズの変化

図21より、ウィンドウサイズがある程度大きくなると、再送タイムアウトを引き起こし、ウィンドウサイズを1に戻す様子が見られる。この実験では、1フローしかデータ転送を行っていないが、マルチフローになった場合、より再送タイムアウトを起こすことが予想される。このように、ウィンドウサイズの増加が再送タイムアウトを引き起こすなら、ある値で制限をかけた方がスループットが良い結果となる。

そこで、提案手法では、再送タイムアウト時のウィンドウサイズの大きさを常に記録しておき、その平均ウィンドウサイズ計算(*ave_window*)し、ウィンドウサイズを増加させる際に、増加後のウィンドウサイズが *ave_window* の 1.5 倍以上になる場合、現在のウィンドウサイズ(*cwnd*)を *cwnd* の 0.75 倍にする制御を加えた。そのアルゴリズムを式(5.3)に示す。

$$\begin{aligned} &if\ (cwnd \geq 1.5 \times ave\_window)\{ \\ &\quad cwnd = 0.75 \times cwnd \\ &\} \end{aligned} \tag{5.3}$$

この制御を加えることにより、ウィンドウサイズをある程度小さな値に制限することができる。そのため、無駄なパケットロス、再送タイムアウトを軽減でき、結果的にスループットの改善にもつながることが期待される。

第 6 章　シミュレーション実験

6.1.　ns2[3,22]

本研究では、提案手法評価のためのシミュレーション実験のネットワークシミュレータとして ns2 を使用した。使用したバージョンは 2.31 である。

ns2 とは、カリフォルニア大学バークレイ校で開発されたネットワークシミュレータのことである。

現在、ns2 はネットワーク技術を研究するための基本ツールとして、世界中の多くの研究者に認知されている。TCP 輻輳制御方式、QoS 制御方式、無線通信ネットワーク、センサネットワーク、マルチエージェントシステムなどに関する研究論文のほとんどは ns2 を用いて実験検証が行われている。さらに、ネットワーク関連の教育補助ツールとして、欧米をはじめ、世界の一部の国の大学、教育研究機構で利用されるようになっている。

ns2 は、C++と Otcl で書かれたオブジェクト指向のイベントドリブン・ネットワークシミュレータであり、ローカル、広域なネットワークをシミュレートするのに有効である。既存のモジュールを組み合わせてシミュレーションを行うには、Otcl のスクリプトを作成すればよい。また、新たなモジュールを追加したい場合には、C++で新たなモジュールを作成し、追加する。図 22 には ns2 によるシミュレーション過程を示す。

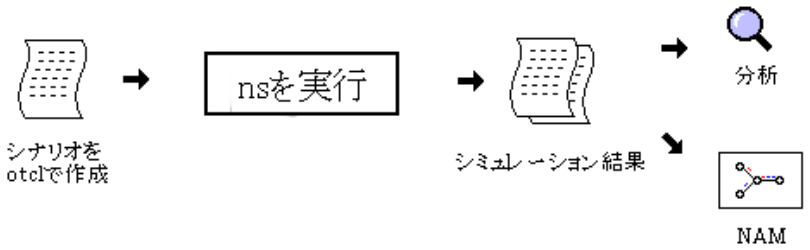

図 22.　ns2 によるシミュレーション過程

6.2.　実験概要

本研究のシミュレーション実験で用いたパラメータを表 4 に示す。Vegas-W のパラメータは図 18 の実験と同様とする。

本研究では、Chain トポロジ、Cross トポロジを用いて、提案手法と従来手法(Reno、Vegas、Ada Vegas、Vegas-W)との比較評価を行った。Chain トポロジを図 23、Cross トポロジを図 24 に示す。

本研究での評価項目として、スループット、パケット到着率、タイムアウト率、ウィン

ドウサイズの変化、時間ごとのスループットの変動など様々な項目で評価を行った。パケット到着率は受信ノードで受信したデータパケット数を送信ノードが送信したデータパケット数で割った値を 100 倍した値とする。再送タイムアウト率は再送タイムアウトで損失したデータパケット数を送信ノードが送信したデータパケット数で割った値を 100 倍した値とする。

表 4.　実験パラメータ

シミュレーション時間	300[sec]
帯域幅	2[Mbps]
ルーティング	AODV
トランスポート	Reno, Vegas, Ada Vegas, Vegas-W, Proposal
MAC	IEEE 802.11
RTS/CTS	ON
キューサイズ	50[pkt]
パケットサイズ	1024[Byte]
エラーレート	1[%]
α, γ	1
β	3
p	1/8
フロー数	N

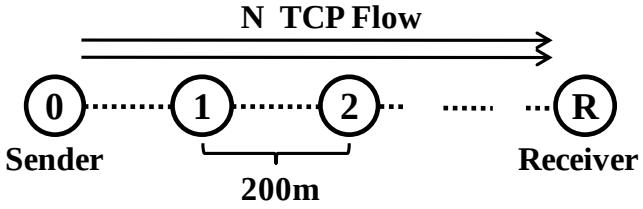

図 23.　Chain トポロジ

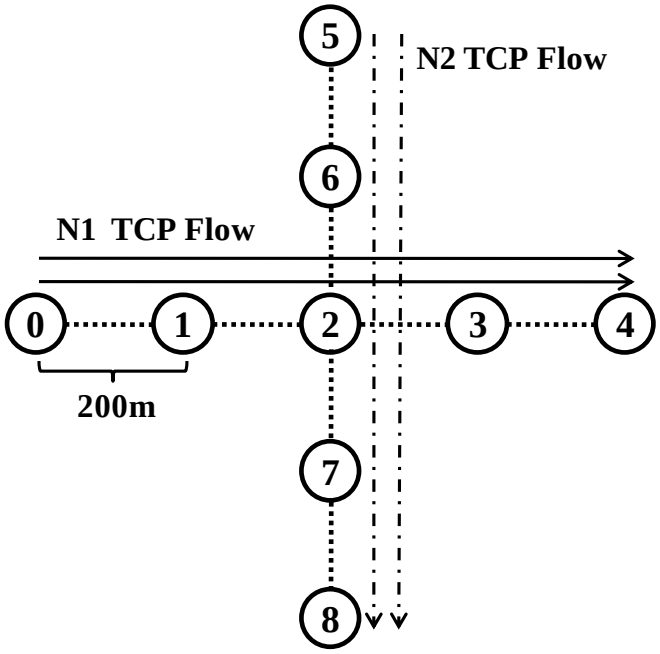

図 24.　Cross トポロジ

6.3.　実験結果

実験結果として、6.3.1.に Chain トポロジでの実験結果、6.3.2.に Cross トポロジでの実験結果を示す。

6.3.1.　Chain トポロジ

図 23 の Chain トポロジを用いて実験を行った。フロー数の N は 1 フロー、2 フロー、4 フローでの評価を行った。

6.3.1.1.　スループット、パケット到着率、再送タイムアウト率特性

ホップ数ごとのスループットの結果、パケット到着率の結果をそれぞれ示していく。1 フロー時のスループット結果を図 25-a、2 フロー時を 25-b、4 フロー時を 25-c に示す。また、1 フロー時のパケット到着率の結果を図 26-a、2 フロー時を 26-b、4 フロー時を 26-c に示す。1 フロー時の Vegas、Ada Vegas、Vegas-W、提案手法の再送タイムアウト率の結果を図 27-a、2 フロー時を図 27-b、4 フロー時を図 27-c に示す。

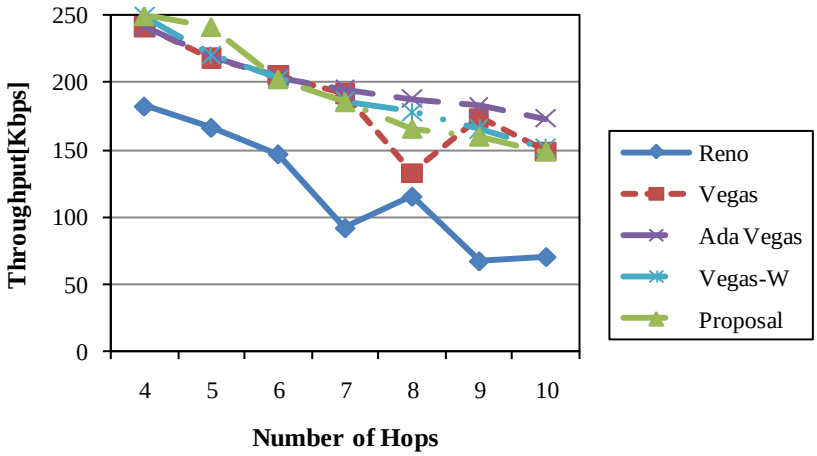

図 25-a.　スループット結果(1flow)

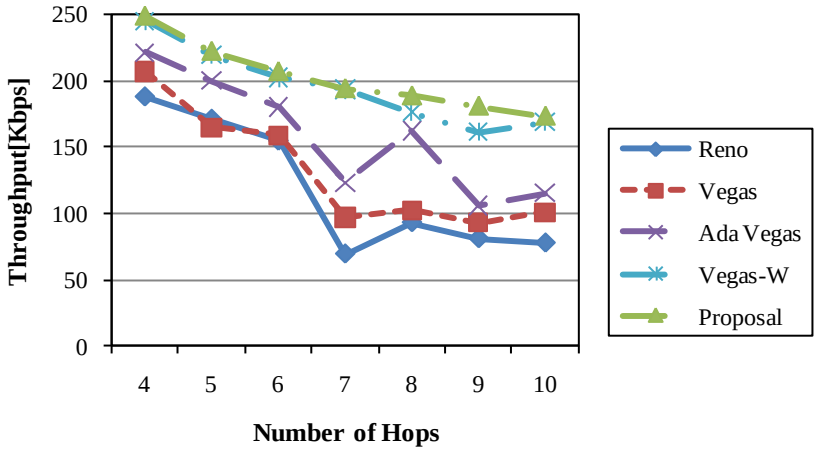

図 25-b.　スループット結果(2flow)

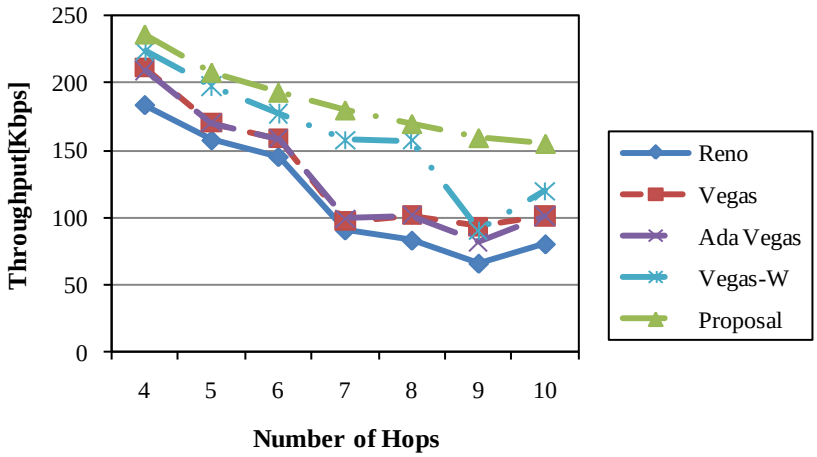

図 25-c.　スループット結果(4flow)

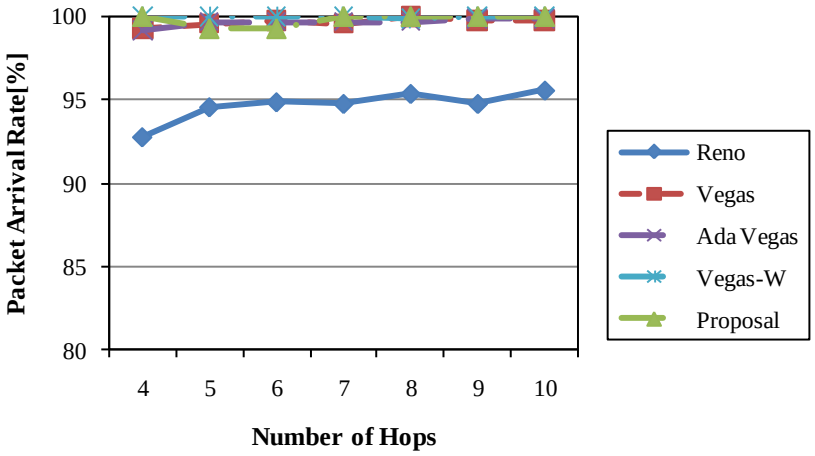

図 26-a.　パケット到着率結果(1flow)

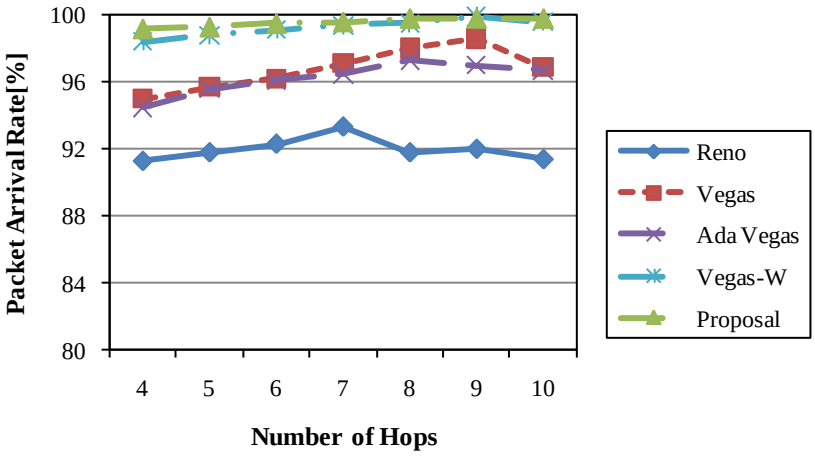

図 26-b.　パケット到着率結果(2flow)

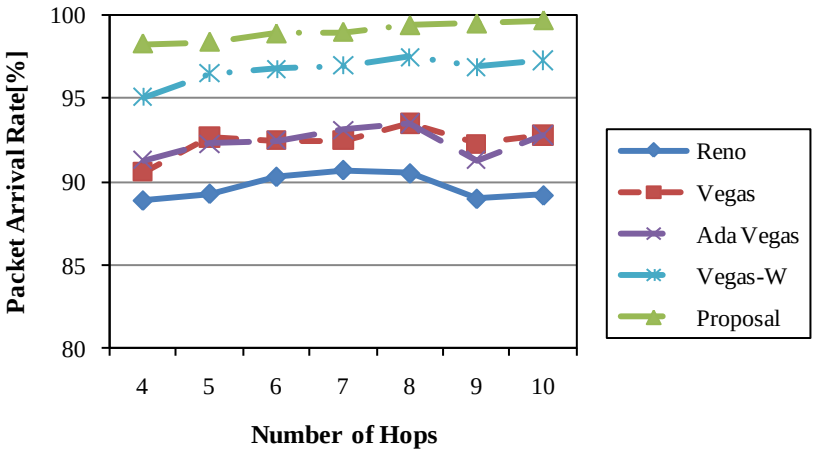

図 26-c.　パケット到着率結果(4flow)

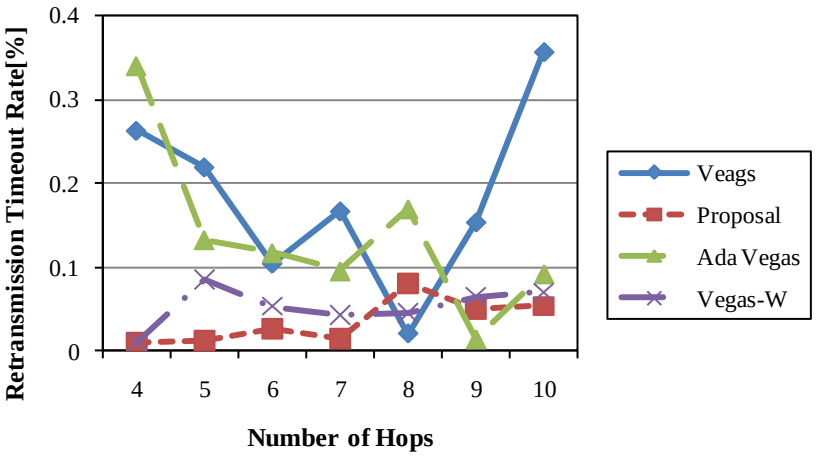

図 27-a.　再送タイムアウト結果(1Flow)

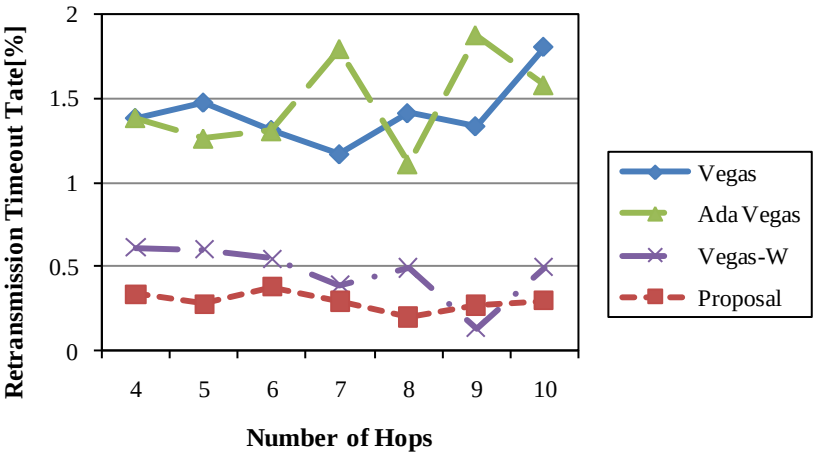

図 27-b.　再送タイムアウト結果(2Flow)

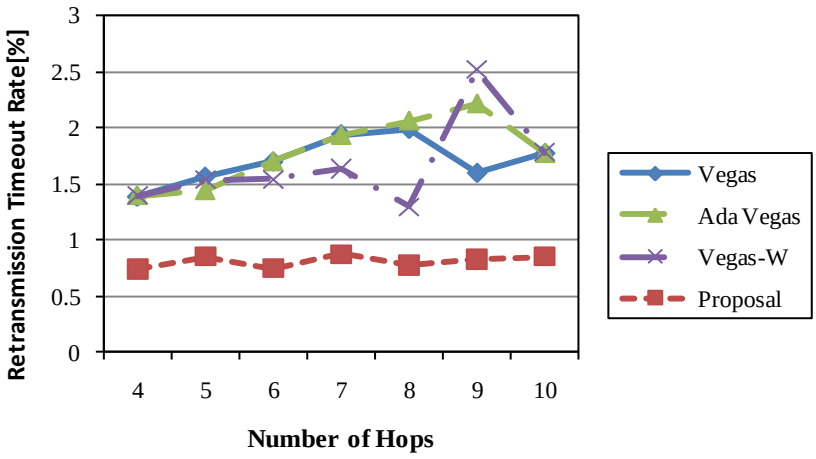

図 27-c.　再送タイムアウト結果(4Flow)

1 フローの場合、提案手法では Vegas とほぼスループットは変わらず、ホップによっては Vegas の方がスループットが良い場合がある。マルチホップアドホックネットワークでの Ada Vegas は *succ* の値が 3 以上になることは少ないため、Vegas と性能はほとんど変わらない。1 フローの場合、他フローとの競合がないため、送信権の競合がない。そのため、MAC 層でのコンテンション、パケット衝突が少ない状況と言える。また、Vegas は RTT を見て、ウィンドウサイズを増減させるため、パケットロスが少ない 1 フローの状況では安定したスループットを得ることができる。図 26-a を見ても、Vegas、提案手法のパケット到着率の結果はほぼ 100%と言え、送信したパケットのほとんどを正しく受信できている。図 27-a を見ても、再送タイムアウトもほとんど起こさないということが分かり、安定してデータ転送を行えている。

図 25-b、25-c の 2 フロー、4 フローの高フローの場合、提案手法でのスループット改善が大きい。2 フローでは最大で約 50%、4 フローでは最大で約 85%のスループット改善が確認できた。1 フローと異なり、2 フロー、4 フローなどの高ホップの場合、他フローとの競合がある。そのため、送信権の競合が行われる。そのため、MAC 層でのコンテンション、パケット衝突、再送タイムアウトが頻繁に起こる状況だと言える。特に高ホップなれば、ノード数が増えるので、パケットロスをする機会が多くなってしまう。Reno など ACK が来る度に、ウィンドウサイズを増加させる手法だと、ネットワークへの負荷を更にかけてしまうため、スループット、パケット到着率が低下してしまう。提案手法ではウィンドウサイズの増加幅を小さくしている点とウィンドウサイズの上限を制限しているため、過度なウィンドウサイズの増加を抑えることができている。そのため、MAC 層でのコンテンション、パケット衝突を軽減できている。図 26-b、26-c を見ても分かるように、従来手法よりパケット到着率の面で大きく改善できているのが分かる。特に 4 フローのように、ネットワーク負荷が大きい状況の場合に提案手法は有効であることが分かる。また、図 27-b、27-c の再送タイムアウトの結果を見ても、Vegas に比べ提案手法は再送タイムアウトの割合が小さいことが分かる。これも、ウィンドウサイズの増加幅を小さくすることにより、ネットワーク負荷を抑えパケットロスを抑えることができ、安定してデータ転送を行えることができた結果であると言える。

Chain トポロジにおける提案手法は、従来手法と比べ、多フロー高ホップの場合に有効であることが確認できた。ネットワーク負荷が高い環境で、ウィンドウサイズの増加幅を小さくすることでネットワーク負荷を抑えることができ、従来手法と比べ、安定したデータ転送が行えることが確認できた。

6.3.1.2. ウィンドウサイズ、RTT 特性

それぞれのフローごとのウィンドウサイズ、RTT の結果を示す。ここでは、10 ホップ時のウィンドウサイズ、RTT の変化の様子を示していく。1 フロー時のウィンドウサイズの変化の様子を図 28-a、2 フロー時のウィンドウサイズの変化の様子を図 28-b、4 フロー時

の Reno のウィンドウサイズの変化の様子を図 28-c、4 フロー時の Vegas のウィンドウサイズの変化の様子を図 28-d、4 フロー時の Ada Vegas のウィンドウサイズの変化の様子を図 28-e、4 フロー時の Vegas-W のウィンドウサイズの変化の様子を図 28-f、4 フロー時の提案手法のウィンドウサイズの変化の様子を図 28-g とする。また、1 フロー時の RTT の変化の様子を図 29-a、2 フロー時の RTT の変化の様子を図 29-b、4 フロー時の Reno の RTT の変化を図 29-c、4 フロー時の Vegas の RTT の変化の様子を図 29-d、4 フロー時の Ada Vegas の RTT の変化の様子を図 29-e、4 フロー時の Vegas-W の RTT の変化の様子を図 29-f、4 フロー時の提案手法の RTT の変化を図 29-g とする。

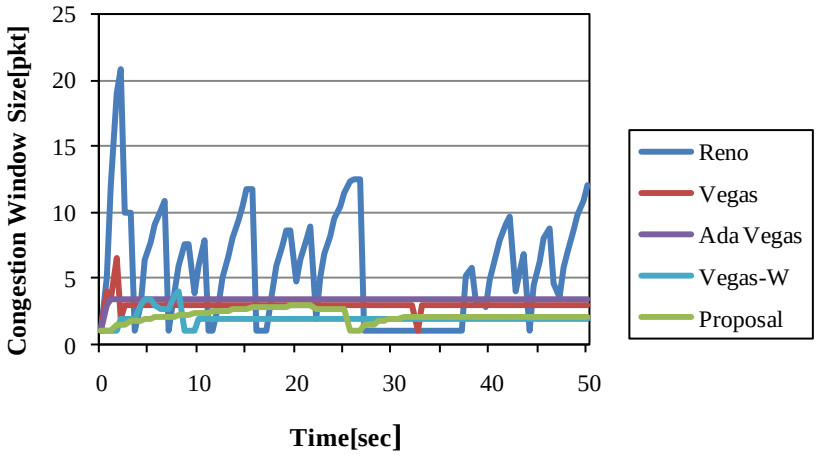

図 28-a.　ウィンドウサイズ変化結果(1Flow)

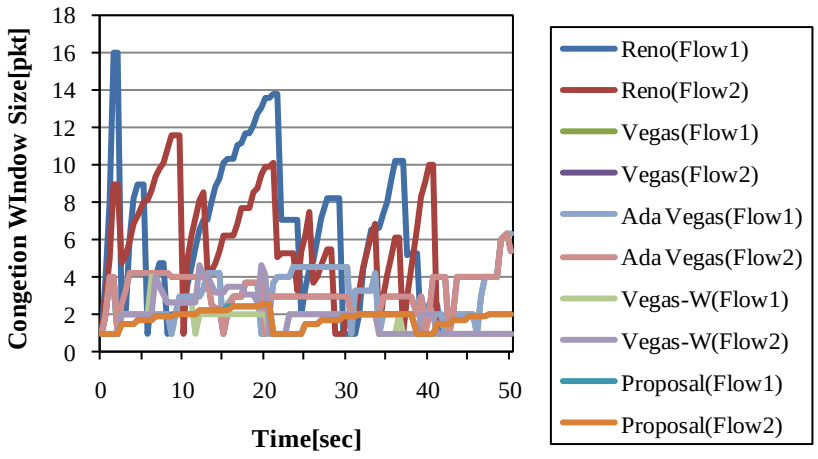

図 28-b.　ウィンドウサイズ変化結果(2Flow)

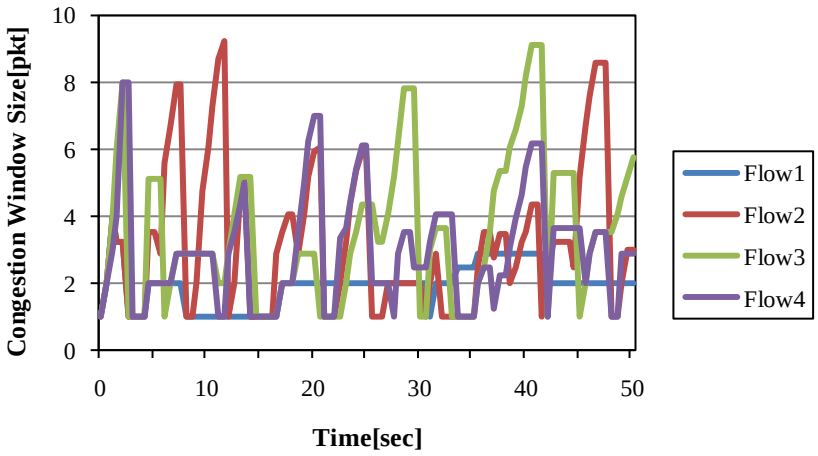

図 28-c.　ウィンドウサイズ変化結果(4Flow,Reno)

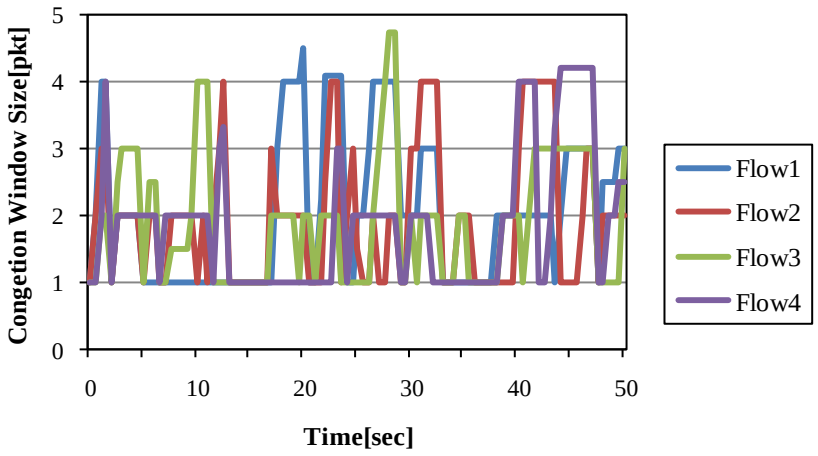

図 28-d.　ウィンドウサイズ変化結果(4Flow,Vegas)

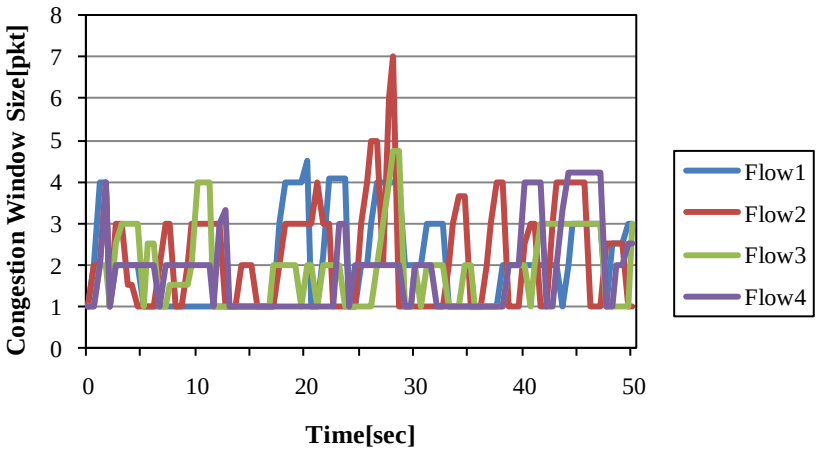

図 28-e.　ウィンドウサイズ変化結果(4Flow,Ada Vegas)

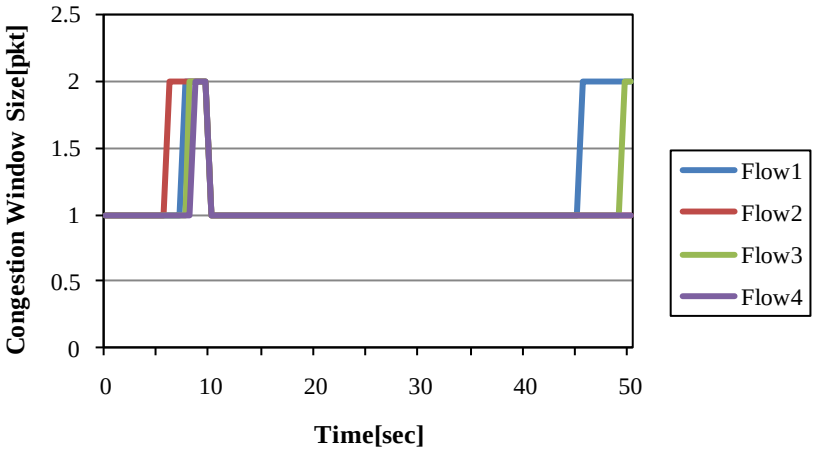

図 28-f.　ウィンドウサイズ変化結果(4Flow,Vegas-W)

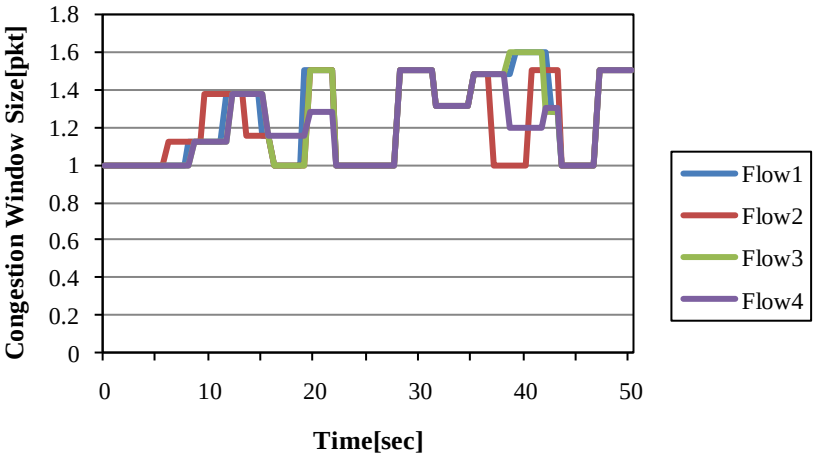

図 28-g.　ウィンドウサイズ変化結果(4Flow,Proposal)

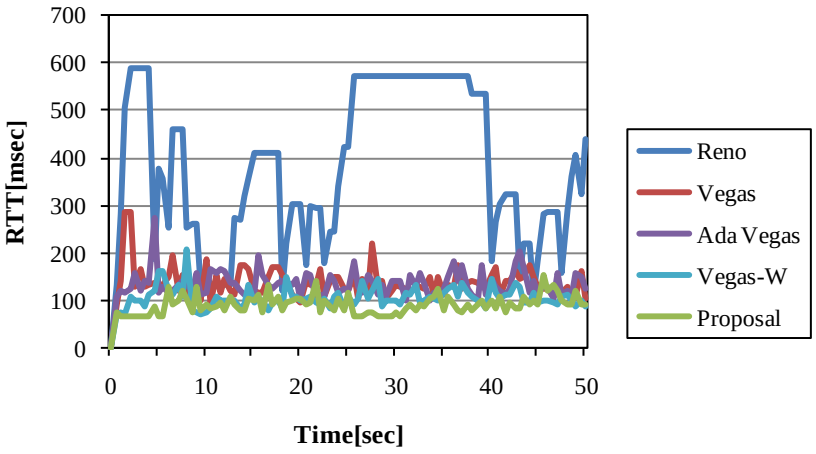

図 29-a.　RTT 変化結果(1Flow)

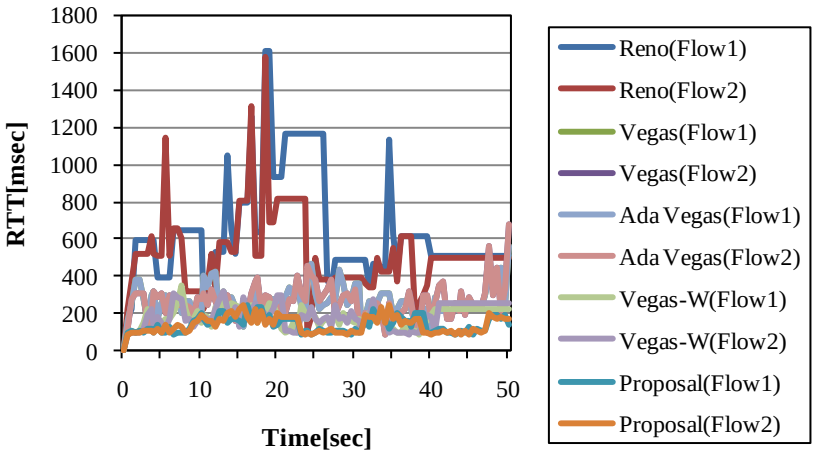

図 29-b.　RTT 変化結果(2Flow)

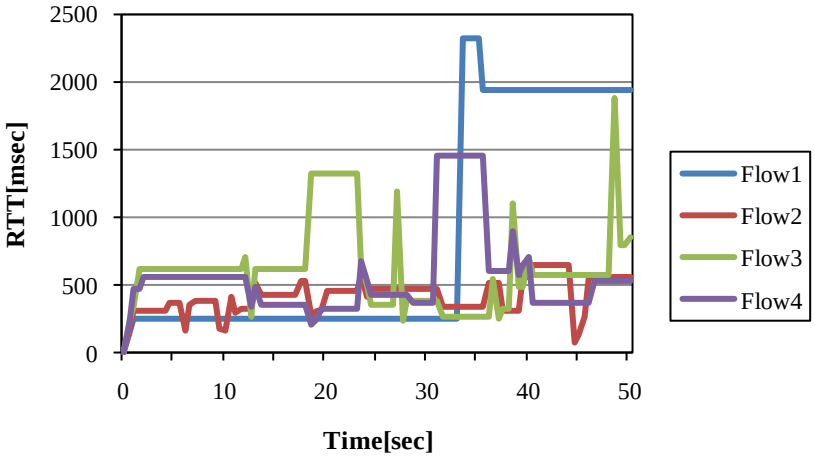

図 29-c.　RTT 変化結果(4Flow,Reno)

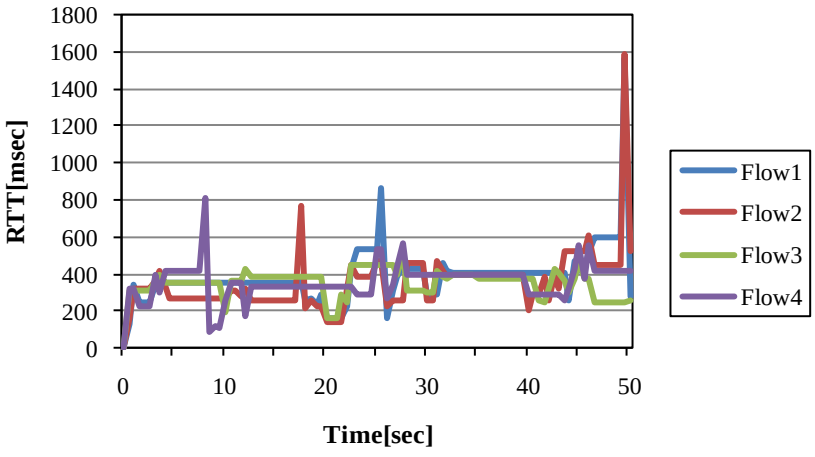

図 29-d.　RTT 変化結果(4Flow,Vegas)

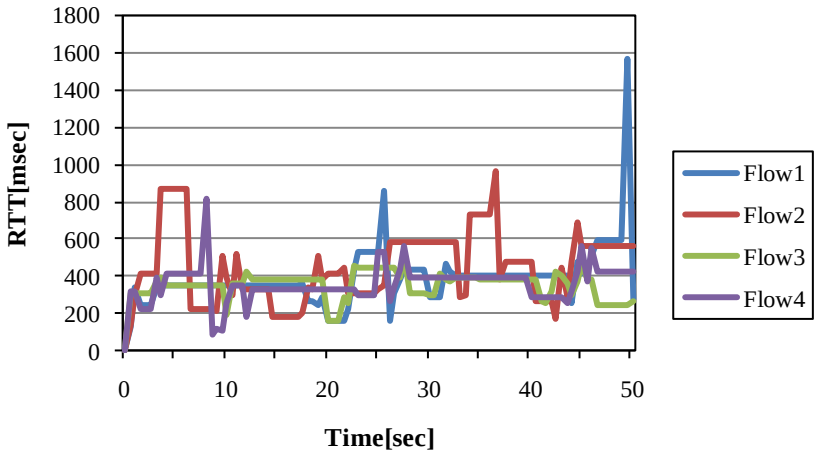

図 29-e.　RTT 変化結果(4Flow,Ada Vegas)

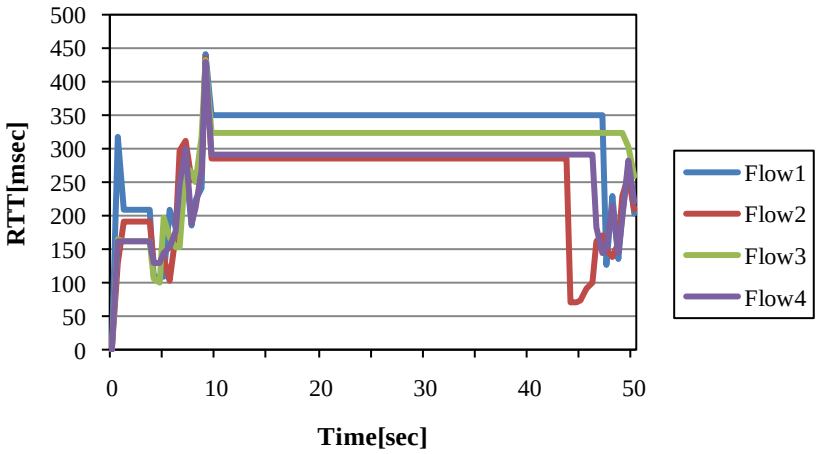

図 29-f.　RTT 変化結果(4Flow,Vegas-W)

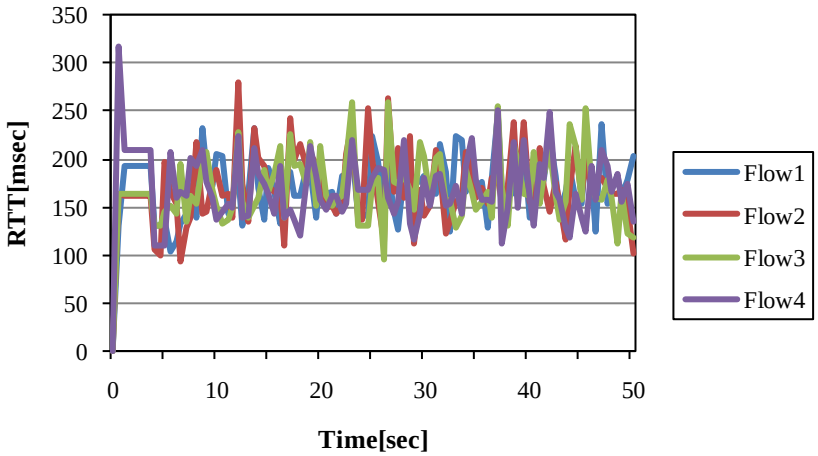

図 29-g.　RTT 変化結果(4Flow,Proposal)

図 28-a の 1 フローの場合のウィンドウサイズの変化の様子を見ると、Vegas、Ada Vegas、Vegas-W、提案手法では、ほとんど再送タイムアウト、パケットロスなく滑らかにウィンドウサイズを増加させ、ある一定値に収束しているのが分かる。1 フローの場合は、他フローとの競合がないため、Vegas、Ada Vegas、Vegas-W、提案手法は安定したウィンドウサイズを得ることができる。そのため、パケット到着率も高い。しかし、Reno ではウィンドウサイズを激しく増減させ、何度も再送タイムアウト、重複 ACK を起こしている。他フローとの競合がないにも関わらず、Reno はウィンドウサイズの過度による増加のため、MAC 層でのコンテンション、パケット衝突等を多く引き起こしているため、再送タイムアウト等を起こしてしまう。また、図 29-a を見ると、ウィンドウサイズの増加に伴い、RTT も増加しているのが分かる。Vegas、Ada Vegas、Vegas-W、提案手法では、ウィンドウサイズがある一定の値に収束するため、RTT の変動も少ないが、Reno ではウィンドウサイズを増減が激しいため、RTT の増減も激しくなる。また、Vegas、Ada Vegas、Vegas-W、提案手法に比べて RTT も大きい。RTT の変動が大きいほど、再送タイムアウト時間(RTO)が大きくなり、パケットを送信するまでの待ち時間が長くなってしまう。そのため、スループットが低下してしまう。図 25-a を見ても、Reno は他の手法に比べ、スループットが低下しているのも分かる。同様にパケット到着率も低下してしまう。

図 28-b の 2 フローの場合は、1 フローの場合と比べ、他フローとの競合があるためパケットロスが起こりやすい環境と言える。Reno は 1 フローの場合と同様にウィンドウサイズを激しく増減させ、図 29-b より RTT の変動も大きいことが分かる。1 フローの場合、安定していた Vegas、Ada Vegas のウィンドウサイズも他フローが入ると、頻繁に再送タイムアウトを引き起こしていることが分かる。1 フローの場合は図 26-a より Vegas、Ada Vegas と Vegas-W、提案手法のパケット到着率はほとんど差がなかったが、2 フロー時の図 26-b を見ると、Vegas、Ada Vegas のパケット到着率が提案手法より小さくなっているのが分かる。それにより、RTT の変動も 1 フローの場合と比べ、大きくなっている。しかし、提案手法では Flow1 と Flow2 が同様のウィンドウサイズの増加の仕方をしており、なめらかにウィンドウサイズを増加させている様子が分かる。再送タイムアウト等のパケットロスも少なく安定したデータ転送ができている。RTT を見ても、増減は少なく Vegas、Ada Vegas より小さい値をとっている。もし、再送タイムアウトが起こっても RTT が小さい場合は、RTO も小さくなり、パケット送信の待ち時間も短くなる。そのため、スループットも大きく低下することはない。図 25-b を見ても、提案手法が他の手法と比べ、スループットの面で大きな値をとっている。

図 28-c、28-d、28-e、28-d、28-g の 4 フローの場合、2 フローより更に、他フローとの競合があるため、2 フローよりパケットロスが起こりやすい環境と言える。図 28-c の Reno ではウィンドウサイズの過度な増加のしすぎで、2 フロー時よりタイムアウト、重複 ACK を引き起こしているのが分かる。そのため、図 29-c を見ても、RTT が最大 2400[msec]まで上がっているのが分かる。そのため、再送タイムアウトが起こった場合、RTO も大きく

なってしまう。図 28-d、28-e の Vegas、Ada Vegas も Reno のようにウィンドウサイズを激しく上下させている。そのため、図 29-d、29-e を見ても、RTT が安定していないのが分かる。図 28-g の提案手法を見ると 2 フロー時と比べて、再送タイムアウト等を引き起こしているものの、他の手法と比べてかなり安定したウィンドウサイズの変化を見せている。図 28-f の Vegas-W ではウィンドウサイズの増加幅が 1 であるため、ネットワークへの負荷も高くすぐに再送タイムアウトを引き起こしている。Vegas よりウィンドウサイズの増加幅を小さくした点とウィンドウサイズの上限を設定しているため、かなり小さな値でウィンドウサイズを遷移させているのが分かる。そのため、図 29-g を見ても、RTT の変動は小さく、大抵 100[msec]から 250[msec]の間で安定している。他の手法の RTT と比べ、平均的に RTT も小さいため、再送タイムアウトが起こっても、スループットを大きく低下しない。そのため、図 25-c から提案手法は、Reno、Vegas、Ada Vegas、Vegas-W より高いスループットを得ることができている。また、パケット到着率の面でも改善を行えている。

6.3.1.3. スループット変化特性

それぞれのフローごとのスループットの変化の様子を示す。ここでは、10 ホップでのスループットの変化の様子を示していく。1 フローでのスループットの変化の様子を図 30-a、2 フローでのスループットの変化の様子を図 30-b、4 フロー時の Reno のスループットの変化の様子を図 30-c、4 フロー時の Vegas のスループットの変化の様子を図 30-d、4 フロー時の Ada Vegas のスループットの変化の様子を図 30-e、 4 フロー時の Vegas-W のスループットの変化の様子を図 30-f、4 フロー時の提案手法のスループットの変化の様子を図 30-g とする。

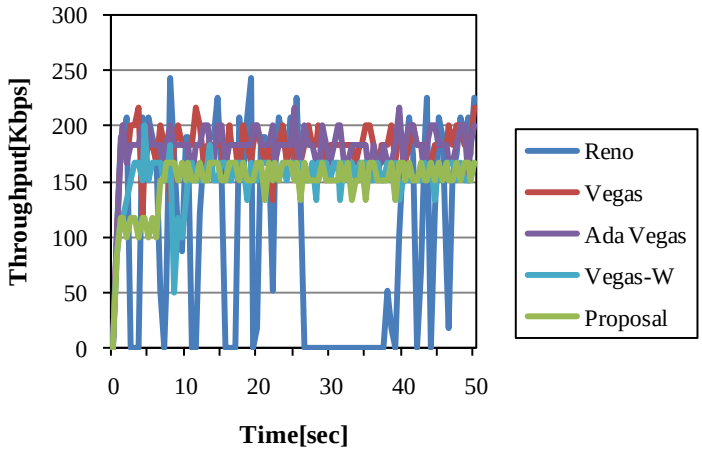

図 30-a. スループット変化結果(1Flow)

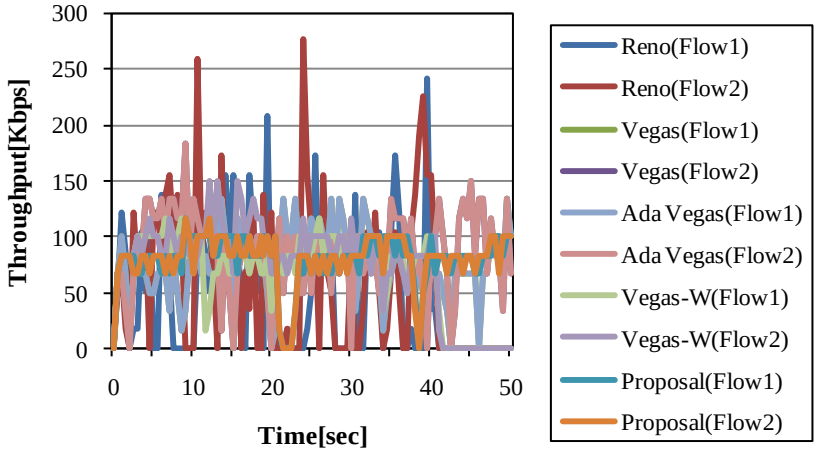

図 30-b.　スループット変化結果(2Flow)

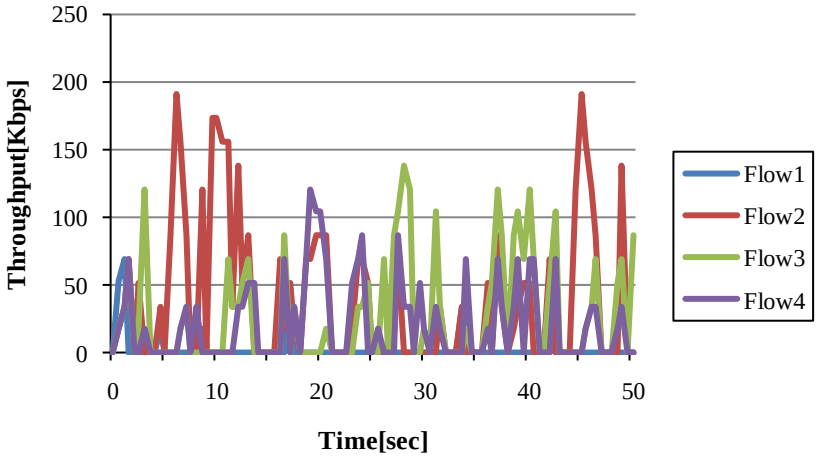

図 30-c.　スループット変化結果(4Flow,Reno)

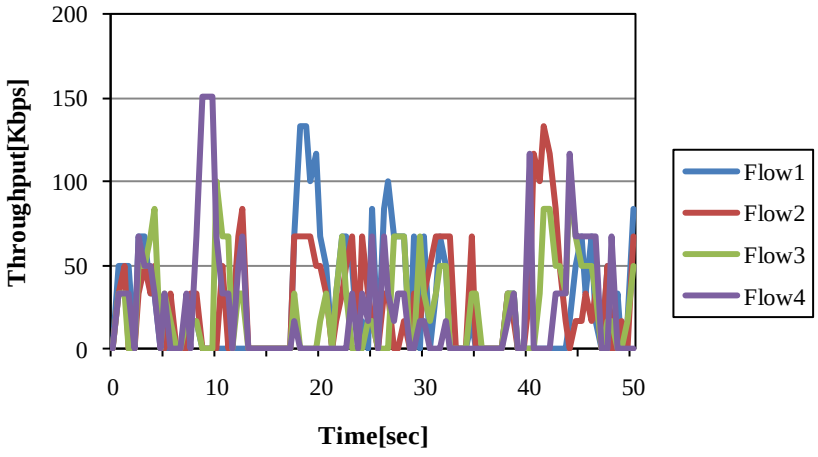

図 30-d.　スループット変化結果(4Flow,Vegas)

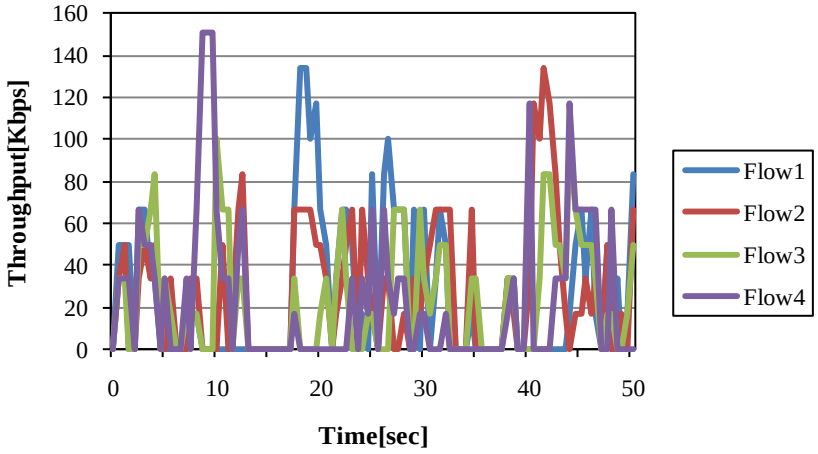

図 30-e.　スループット変化結果(4Flow,Ada Vegas)

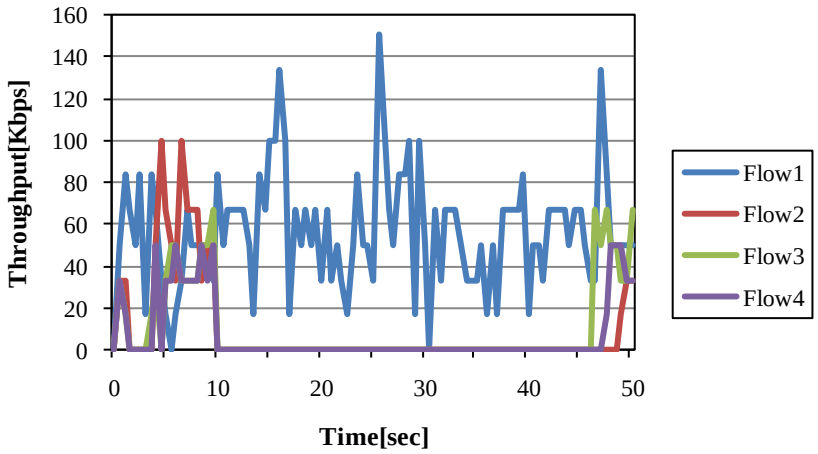

図 30-f.　スループット変化結果(4Flow,Vegas-W)

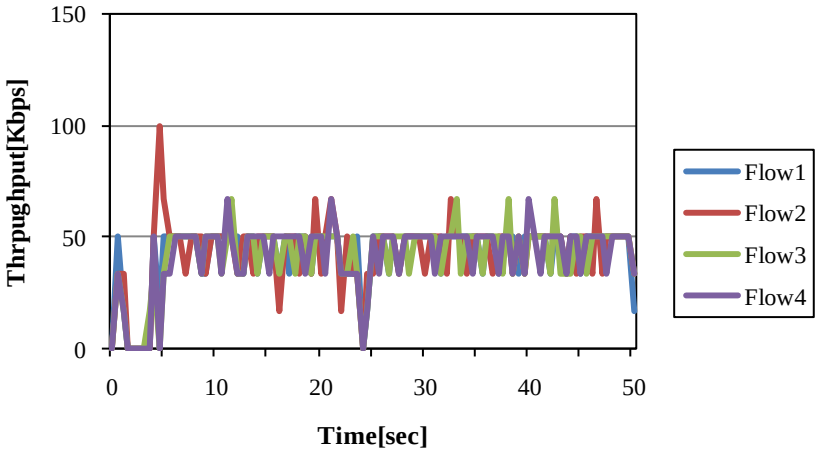

図 30-e.　スループット変化結果(4Flow,Proposal)

図 30-a の 1 フローの場合のスループット変化の様子を見ると、Vegas、Ada Vegas、Vegas-W、提案手法が安定してスループットの変化をしているのが分かる。これは、1 フローの場合は他フローとの競合がないため、パケットのロスが起こりにくいため、Vegas、Ada Vegas、Vegas-W、提案手法は安定したスループットを得ることができる。

図 30-b の 2 フローの場合のスループット変化の様子を見ると、Reno ではウィンドウサイズを激しく増減させているため、スループットも安定せず、増減が激しい。1 フロー時に安定した Vegas、Ada Vegas のスループットも、他フローとの競合が起こると、図 28-b からウィンドウサイズを頻繁に 1 に戻しているのが分かる。そのため、1 フローに時に比べ、スループットも安定しないで、激しく増減させている。しかし、Vegas-W、提案手法では、再送タイムアウトをあまり、起こさず、安定してウィンドウサイズを変化させることができているので、スループットも安定して変化している様子が分かる。

図 30-c、30-d、30-e、30-f、30-g の 4 フローの場合のスループットの変化の様子をみると、2 フローの場合と比べ、更にスループットの増減が激しくなっているのが分かる。これは図 28-c、28-d、28-e、28-f、28-g よりウィンドウサイズを再送タイムアウト等のパケットロスのため、何度も 1 に戻しているため、スループットも激しく上下に変動してしまうためである。Reno ではウィンドウサイズを積極的に上げるため、1 回の送信量が大きい。そのため、Vegas、Ada Vegas、Vegas-W、提案手法と比べ、スループットの最大値が高い。Vegas、Ada Vegas もウィンドウサイズの変化の様子から何度も再送タイムアウトを引き起こしているので、スループットが安定しない。Vegas-W もウィンドウサイズの増加スピードを遅くはしているものの、何度も再送タイムアウトを引き起こしているのでスループットが安定しない。しかし、提案手法では、再送タイムアウト等のパケットロスが少ないため、安定してウィンドウサイズを増加させることができている。そのため、スループットも安定して変化しているのが分かる。提案手法では、ウィンドウサイズの増加幅を小さくしているため、1 回の送信量がすくない。そのため、Reno、Vegas、Ada Vegas に比べ、スループットの値自体は小さいが、50[Kbps]付近で安定した値を得ることができている。安定してデータ転送を行えているため、トータルのスループットは Reno、Vegas、Ada Vegas、Vegas-W と比べ、大きな値を得ることができている。

スループットの変化の様子を見ても分かるように、スループットが安定するかどうかはウィンドウサイズが安定するかどうかで決定される。多フロー高ホップでは、特に提案手法は安定したウィンドウサイズを得ることができるため、安定したデータ転送を行えることができる。そのため、スループットも安定し、Reno、Vegas、 Ada Vegas、Vegas-W より高いスループットを得ることができている。

6.3.2. Cross トポロジ

図 24 の Cross トポロジを用いて実験を行った。フロー数の N は、2 フロー、4 フローでの評価を行った。2 フローの場合は、N1 が 1、N2 も 1 とする。また、4 フローの場合は、

N1 が 2、N2 も 2 とする。

6.3.2.1. スループット、パケット到着率、再送タイムアウト率特性

フローごとのスループットの結果、パケット到着率の結果をそれぞれ示していく。2 フロー時を 31-a、4 フロー時を 31-b に示す。また、2 フロー時のパケット到着率を 32-a、4 フロー時を 32-b に示す。フローごとの Vegas、提案手法の再送タイムアウト率の結果を図 33 に示す。

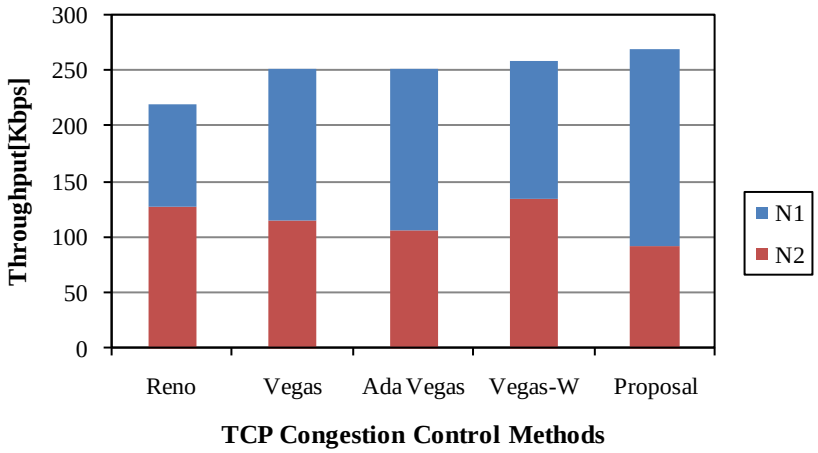

図 31-a. スループット結果(2Flow)

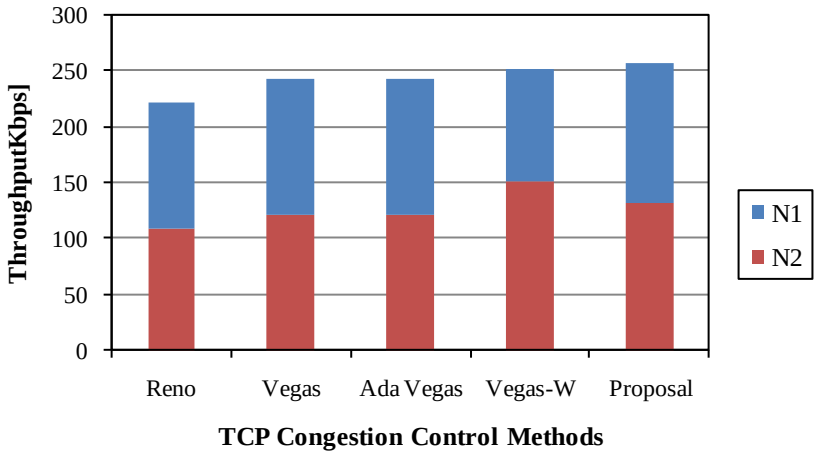

図 31-b. スループット結果(4Flow)

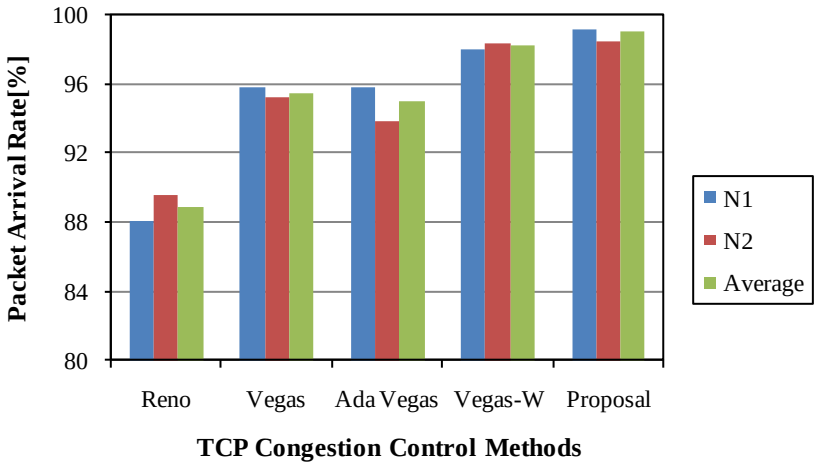

図 32-a.　パケット到着率結果(2Flow)

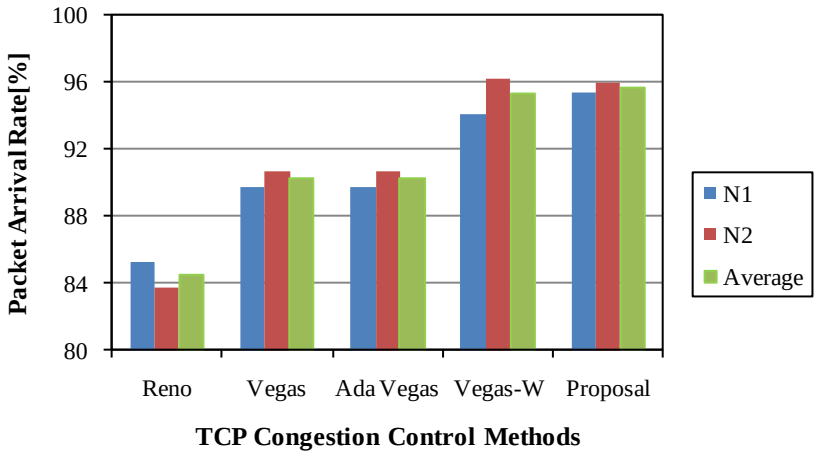

図 32-b.　パケット到着率結果(4Flow)

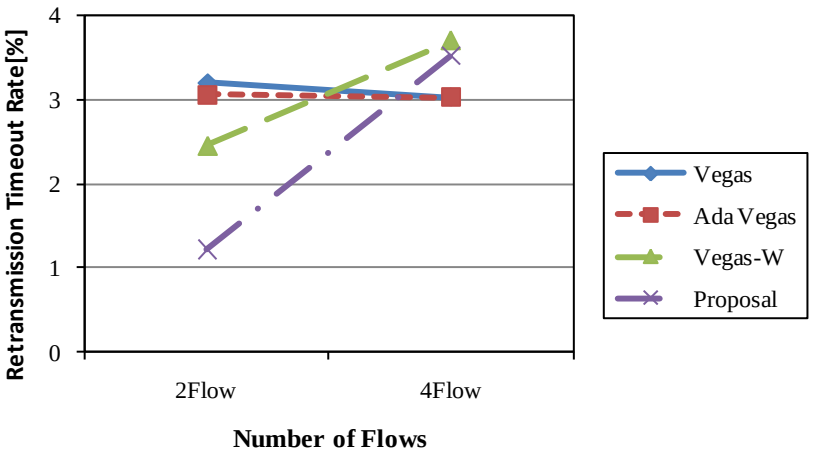

図 33.　再送タイムアウト結果(Cross トポロジ)

図 31-a の 2 フローの場合を見ると、Vegas に比べ、提案手法では約 7%のスループット改善が行えている。Cross トポロジの場合、ノード 2 における MAC 層のコンテンション、パケットロス等が起こりやすい。ノード 2 では、ノード 0 からの横フローとノード 5 からの縦フローが交差する場所である。そのため、ノード 1 とノード 6 でパケット送信権の競合が起こる。そのため、パケット衝突が起こり、結果として再送タイムアウトを引き起こしてしまう。また、ノード 0 とノード 5 がキャリアセンス外にあるため、キャリアセンスを行えずそれぞれがそれぞれのタイミングでパケットを送信してしまう。そのため、コンテンション、パケット衝突を引き起こしてしまう。結果として再送タイムアウトを引き起こしてしまう。また、マルチホップ通信の場合、共有チャネルを使用するため、1 ユーザのみがパケットを送信できる。そのため、横、縦どちらかのフローがパケット送信を成功した場合、そのまま、そのフローが帯域を占有してしまい、1 フロー4 ホップの Chain トポロジと同様な通信形態となってしまう。特に提案手法では再送タイムアウト等のロスを起こりにくくしているため、片方のフローが占有してしまうことが確認できた。結果から N1 フローが大きなスループットを得ている。図 25-a から 1 フロー4 ホップでは Vegas と提案手法でスループットの差がほとんどない。そのため、Cross トポロジの 2 フローでは提案手法の改善効果が小さい。パケット到着率の面では図 32-a から分かるように、提案手法は Reno、Vegas、Ada Vegas、Vegas-W と比べて改善できている。提案手法では、ウィンドウサイズの増加幅を小さくする点とウィンドウサイズの上限を設定しているため、ネットワーク負荷を抑えることができ、その結果パケット到着率を改善することが可能である。しかし、Cross トポロジでは Chain トポロジと比べ、パケットがロスしやすい環境であるため、Chain トポロジと比べてパケット到着率は低下してしまう。

図 31-b の 4 フローの場合を見ると、Vegas に比べ、提案手法では約 7%のスループット改善が行えている。4 フローは 2 フローと比べ、更にネットワーク負荷が増え、パケットがロスしやすい環境と言える。2 フロー同様、4 フローでも片方のフローが占有してしまうと、そのフローがパケットロスするまで、もう一方のフローがパケット送信することが難しい。しかし、2 フローの場合と比べ、パケットがロスしやすいため、片方のフローが占有するといったことは少ない。パケット到着率の結果を見ても、Chain トポロジの場合と比べて大きく低下してしまっている。これは多フローとの競合が 2 フローの場合より、パケットの送信権の競合が厳しくなったためである。そのため、図 33 から 2 フロー時は再送タイムアウトの面で改善できているが、4 フローの場合は Vegas と提案手法と変わらない結果となった。特にノード 0 とノード 5 がキャリアセンス外にあるため、それぞれのタイミングでパケットを送信することで再送タイムアウトを引き起こしてしまうことが原因である。

6.3.2.2. ウィンドウサイズ、RTT 特性

図 24 において、2 フロー時の N1 をフロー1、N2 をフロー2 とする。また、4 フロー時の N1 をフロー1、フロー2、N2 をフロー3、フロー4 とする。

それぞれのフローごとのウィンドウサイズ、RTT の結果を示す。2 フローでのウィンドウサイズの変化の様子を図 34-a、4 フローでの Reno のウィンドウサイズの変化の様子を図 34-b、4 フロー時の Vegas のウィンドウサイズの変化の様子を図 34-c、4 フロー時の Ada Vegas のウィンドウサイズの変化の様子を図 34-d、4 フロー時の Vegas-W のウィンドウサイズの変化の様子を図 34-e、4 フロー時の提案手法のウィンドウサイズの変化の様子を図 34-f とする。また、2 フローでの RTT の変化の様子を図 35-a、4 フローでの Reno の RTT の変化の様子を図 35-b、4 フロー時の Vegas の RTT の変化の様子を図 35-c、4 フロー時の Ada Vegas の RTT の変化の様子を図 35-d、4 フロー時の Vegas-W の RTT の変化の様子を図 35-e、4 フロー時の提案手法の RTT の変化の様子を図 35-f とする。

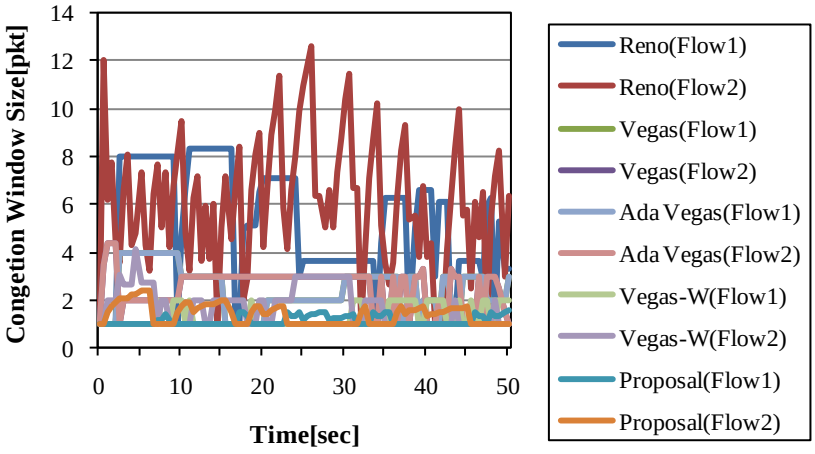

図 34-a.　ウィンドウサイズ変化結果(2Flow)

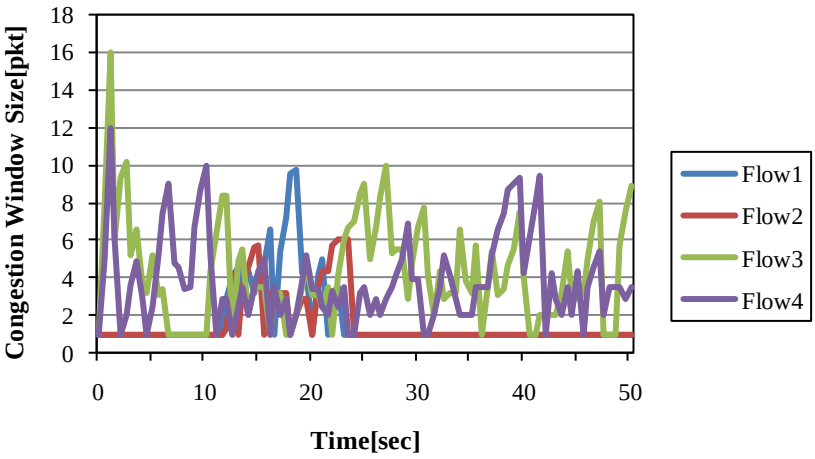

図 34-b.　ウィンドウサイズ変化結果(4Flow,Reno)

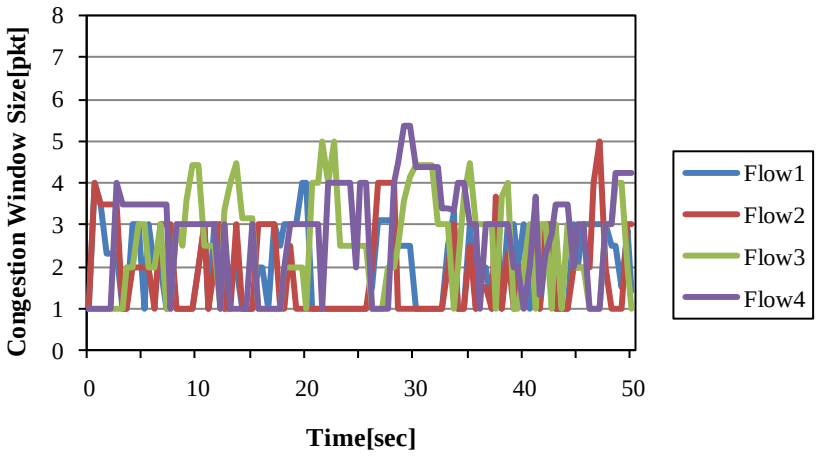

図 34-c.　ウィンドウサイズ変化結果(4Flow,Vegas)

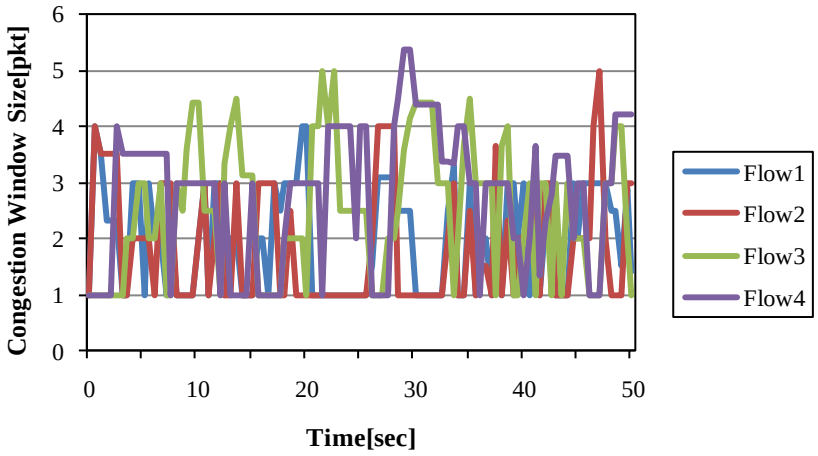

図 34-d.　ウィンドウサイズ変化結果(4Flow,Ada Vegas)

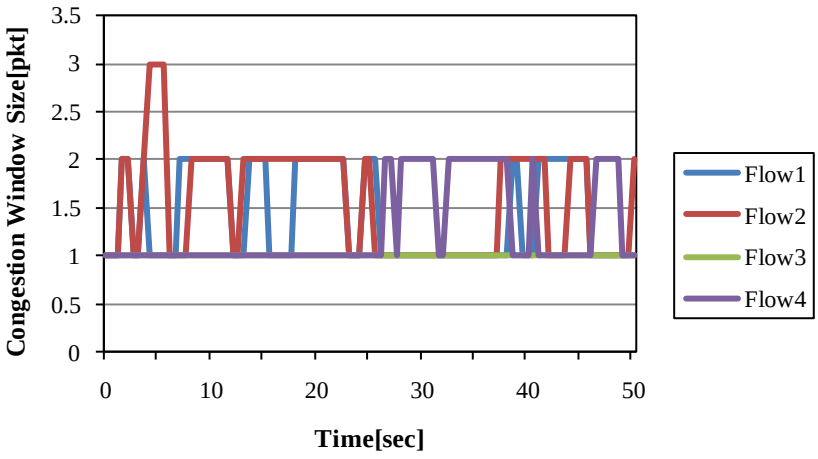

図 34-e.　ウィンドウサイズ変化結果(4Flow,Vegas-W)

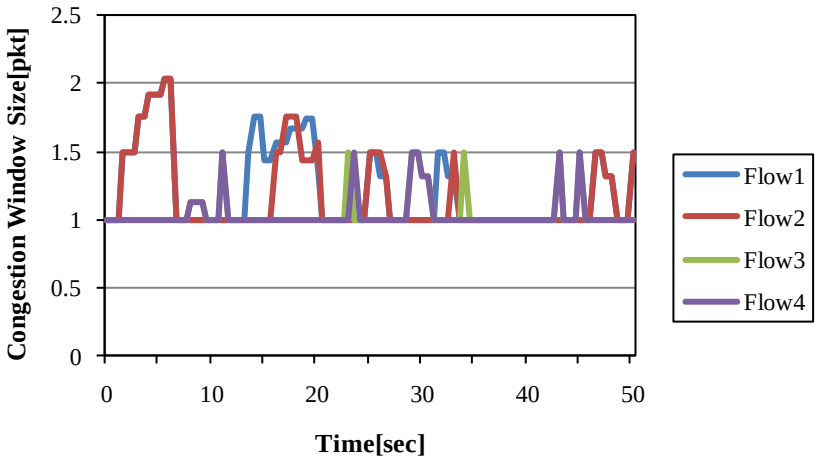

図 34-f.　ウィンドウサイズ変化結果(4Flow,Proposal)

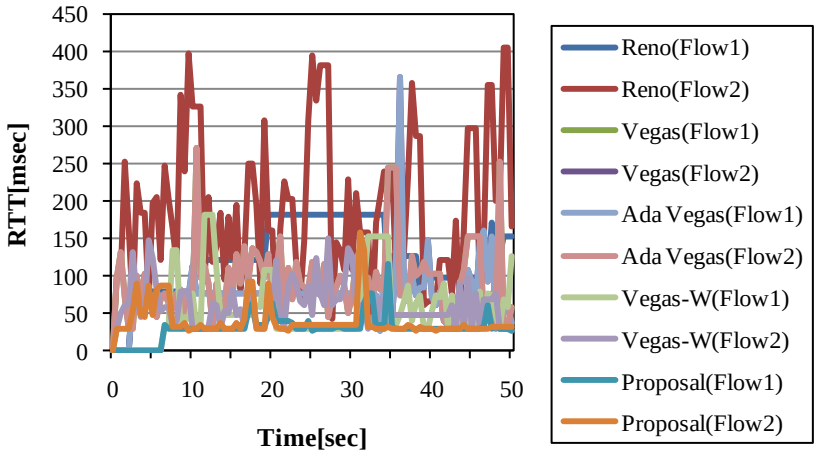

図 35-a.　RTT 変化結果(2Flow)

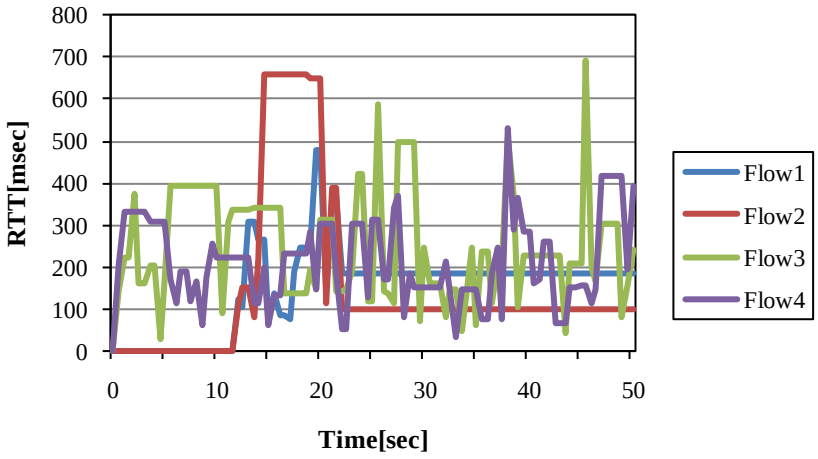

図 35-b.　RTT 変化結果(4Flow,Reno)

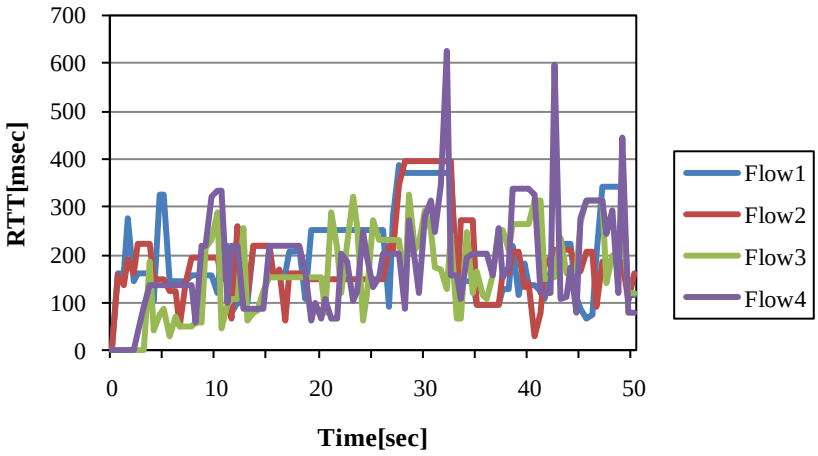

図 35-c.　RTT 変化結果(4Flow,Vegas)

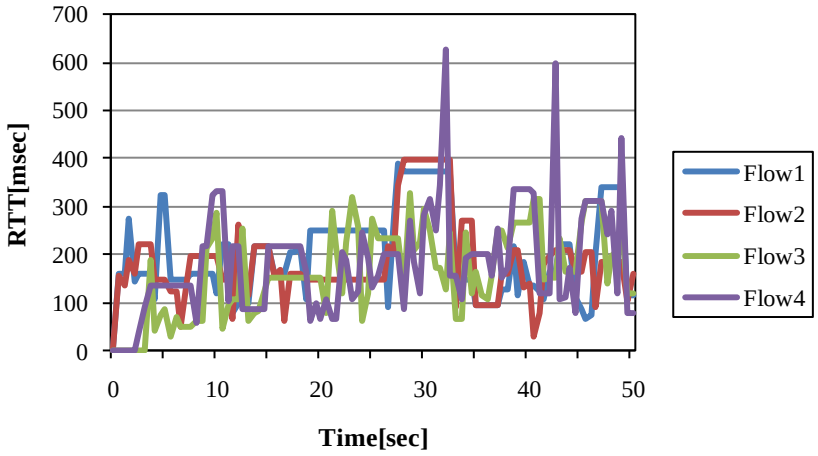

図 35-d.　RTT 変化結果(4Flow,Ada Vegas)

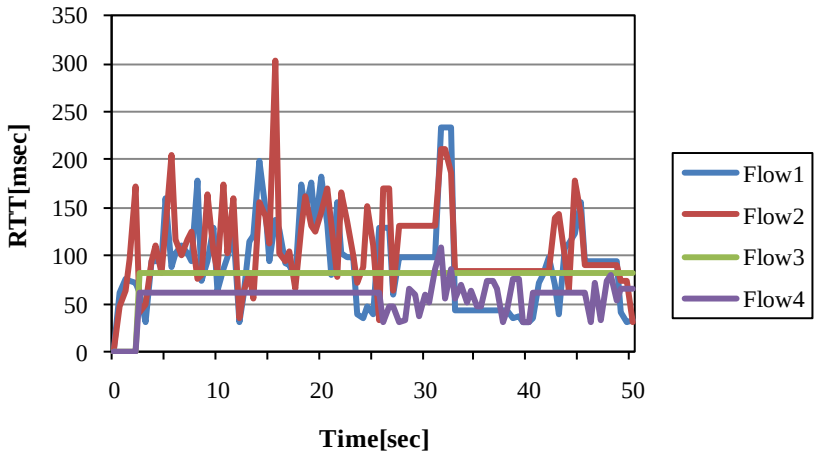

図 35-e.　RTT 変化結果(4Flow,Vegas-W)

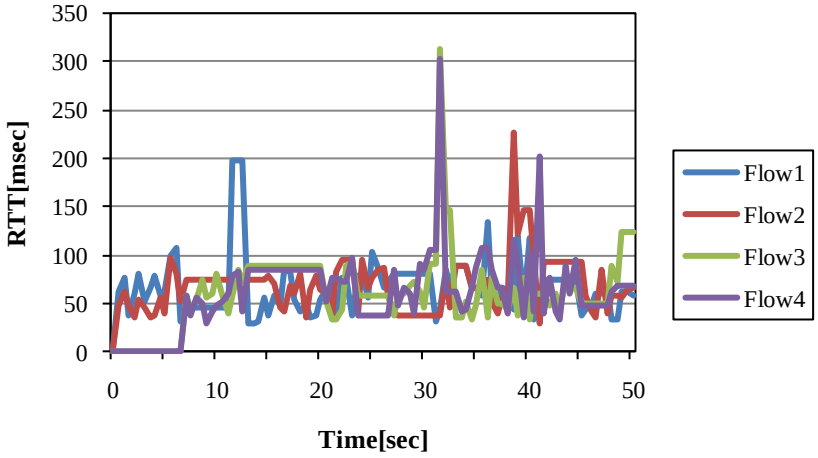

図 35-f.　RTT 変化結果(4Flow,Proposal)

図 34-a の 2 フローの場合のウィンドウサイズの変化の様子を見ると、どの輻輳制御方式にも共通しているのは、フロー1 かフロー 2 のどちらかがウィンドウサイズを増加させている場合に、もう片方のフローはウィンドウサイズを増加させることができない点である。これは、マルチホップ通信では無線チャネルを共有しているため、1 回に送信できるのは 1 ユーザのみである。そのため、1 回パケット送信を成功させるとそのフローが帯域を占有してしまう問題がある。Reno はフロー2 がウィンドウサイズを激しく、増減させており、フロー1 がほとんど通信できていないのが分かる。図 35-a を見ても、フロー2 の RTT の変動が大きい。Vegas、Ada Vegas、Vegas-W では、Reno に比べ安定してパケットを送信できており、提案手法とほぼ同様な動作をしている。RTT の変動を見ても、Vegas、Ada Vegas、Vegas-W と提案手法で大きな差はないが、Vegas、Ada Vegas、Vegas-W ではウィンドウサイズを 1 増加させる点でネットワーク負荷が大きく、全体的に提案手法より RTT が大きい値を取っている。

図 34-b の 4 フローの場合の Reno のウィンドウサイズの変化の様子を見ると、初めの 10 秒程度までフロー3 とフロー4 の N2 フローがウィンドウサイズを増加させており、フロー1、フロー2 を増加させることができていない。そのため、N1 フローは全く通信できていない。片方のフローが帯域を占有すると、もう片方のフローが入り混むのは難しい状況と言える。Reno では、2 フロー時同様ウィンドウサイズの増減が激しいため、安定した通信ができていない。図 34-c、34-d、34-e の 4 フローの場合の Vegas、Ada Vegas、Vegas-W のウィンドウサイズの変化の様子を見ると、2 フロー時と比べ、ウィンドウサイズが安定していない。4 フローのようにネットワーク負荷が大きい状態になると、パケットロスが起こりやすい状況であるのと Cross トポロジでは、Chain トポロジと比べ、よりパケットロスが起こりやすい。そのため、Chain トポロジよりウィンドウサイズが安定しない。図 34-f の提案手法を見ると、再送タイムアウトを引き起こすものの、緩やかにウィンドウサイズを

変化させているのが分かる。ウィンドウサイズも Reno、Vegas、Ada Vegas、Vegas-W と比べると、小さいためネットワーク負荷を抑えられ、RTT の変動も小さい。そのため、再送タイムアウトを起こしてもパケット送信までの待ち時間も少ない。そのため、安定してデータ転送を行えることができている。

6.3.2.3. スループット変化特性

それぞれのフローごとのスループットの変化の様子を示す。2 フロー時のスループットの変化の様子を図 36-a、4 フロー時の Reno のスループットの変化の様子を図 36-b、4 フロー時の Vegas のスループットの変化の様子を図 36-c、4 フロー時の Ada Vegas のスループットの変化の様子を図 36-d、4 フロー時の Vegas-W のスループットの変化の様子を図 36-e、4 フロー時の提案手法のスループットの変化の様子を図 36-f とする。

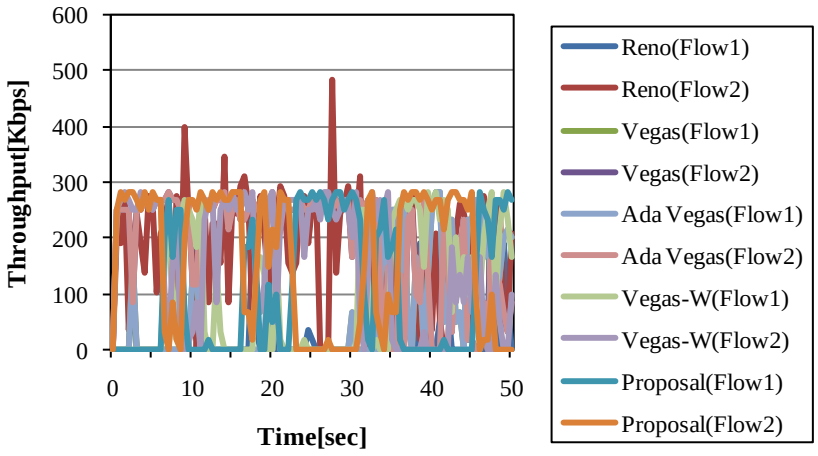

図 36-a. スループット変化結果(2Flow)

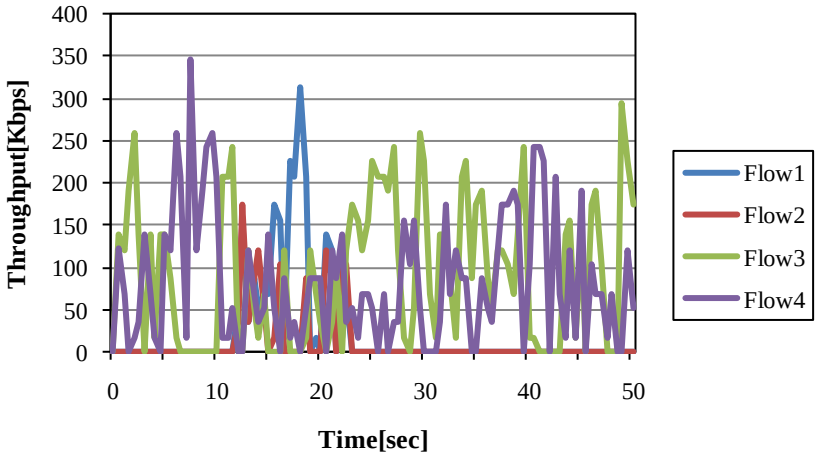

図 36-b. スループット変化結果(4Flow,Reno)

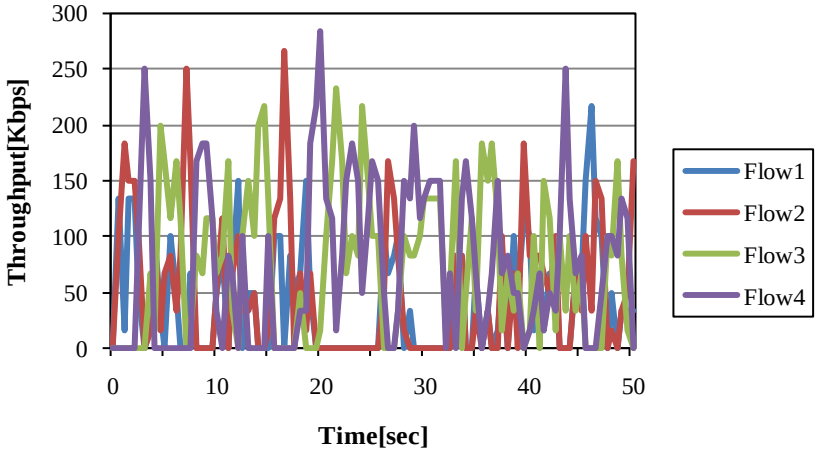

図 36-c.　スループット変化結果(4Flow,Vegas)

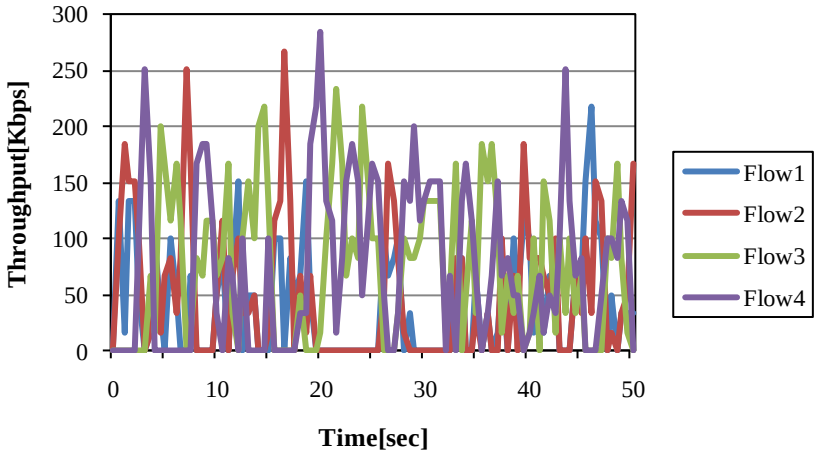

図 36-d.　スループット変化結果(4Flow,Ada Vegas)

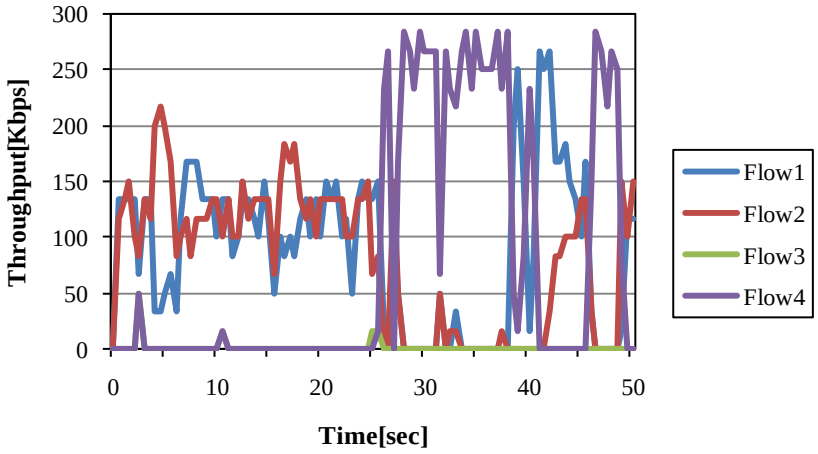

図 36-e.　スループット変化結果(4Flow,Vegas-W)

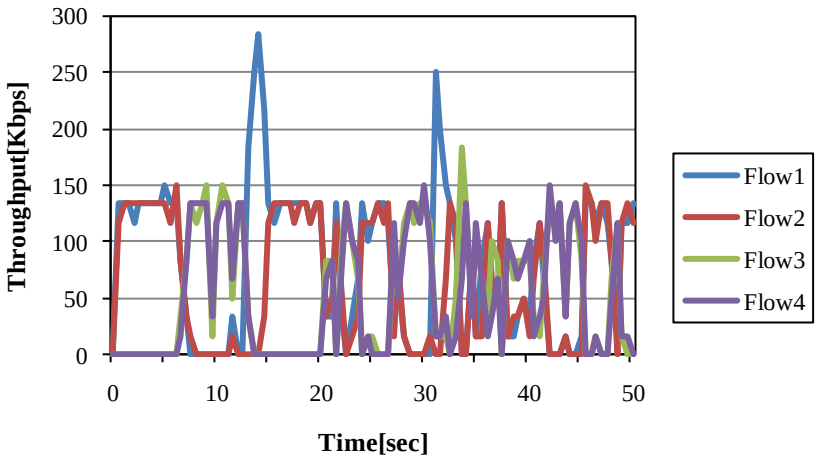

図 36-d.　スループット変化結果(4Flow,Proposal)

図 36-a の 2 フローの場合を見ると、フロー1 が帯域を占有している時、フロー2 はほとんど通信できていないのが分かる。これは図 34-a のウィンドウサイズの結果からも分かる。Reno ではウィンドウサイズを激しく、増減させている分スループットの増減も大きい。Vegas、Ada Vegas、Vegas-W、提案手法では、Reno に比べウィンドウサイズが安定しているため、スループットも安定しているのが分かる。

図 36-b、36-c、36-d、36-e、36-f の 4 フローの場合のスループットの変化の様子をみると、Reno ではフロー3、4 が帯域を占有してしまっているため、フロー1、2 がほとんど通信できていない状況にある。Vegas、Ada Vegas、Vegas-W、提案手法の結果からも同様なことが言える。フロー数が多くなるほど、ネットワーク負荷が大きくなり、MAC 層のコンテンション、パケット衝突、再送タイムアウトが起こりやすくなる。そのため、どの輻輳制御方式においても 2 フローよりかスループットの安定性はなく、スループットの変動が大きい。しかし、Reno、Vegas、Ada Vegas、Vegas-W に比べ、提案手法ではウィンドウサイズの増加幅を小さくする点とウィンドウサイズの上限を設定することで、パケットロスを減少でき、安定したデータ転送が行えていると言える。

第 7　章　結論

7.1.　総括

本研究では、無線マルチホップアドホックネットワークにおける輻輳制御方式を提案した。ns2 によるシミュレーション実験から従来手法(Reno,Vegas,Ada Vegas,Vegas-W)に比べて、スループット、パケット到着率、再送タイムアウト率の面で改善できたことを確認できた。マルチホップ通信では Reno などのウィンドウサイズの増加幅では、アグレッシブすぎてパケットロスを引き起こす原因となってしまう。提案手法ではウィンドウサイズの増加幅を小さくする点とウィンドウサイズの上限を制限することにより、ある程度ウィンドウサイズを小さな値にすることができる。そのため、MAC 層でのコンテンション、衝突などを軽減でき、再送タイムアウトを引き起こさず安定したデータ転送ができる。特に多フロー高ホップなどのネットワーク負荷が高い場合に、提案手法が有効であることが確認できた。

7.2.　今後の課題

今後の課題として以下のようなことが挙げられる。

- α、β、γ などの定数の評価
- 提案手法における *succ*、*count* などの定数の評価
- Grid、Random トポロジでの評価

以上のように、まだ未着手の実験、評価すべき項目がまだ存在する。特に提案手法におけるウィンドウサイズの増加幅に定数を使っているので、高帯域、低帯域の両方に適した手法だとは言えない。高帯域、低帯域の両方に適した輻輳制御方式にするため、RTT などの動的パラメータを考慮した輻輳制御方式を提案、開発する必要がある。

参考文献

[1] Z.Fu, P.Zerfos, H.Luo, S.Lu, L.Zhang, M.Gerla, “The Impact of Multihop Wireless Channel on TCP Throughput and Loss”, IEEE INFCOM 2003, Vol.3 pp.1744-1753, 2003.

[2] L.S.Brakmo, L.L.Perterson, “TCP Vegas: End-to-End Congestion Avoidance on a Global Internet”, IEEE Journal on Selected Areas in Communication, Vol.13, August 1995.

[3] “The Network Simulator –ns-2–“, http://www.isi.edu/nsnam/ns/

[4] C-K.Toh, “アドホックモバイルワイヤレスネットワーク　ープロトコルとシステー”, 共立出版, 2003.

[5] C.E.Perkins, P.Bhagwat, “Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers”, ACM SIGCOMM 1994, pp.234-244, 1994.

[6] C.Perkins, E.Belding-Royer, S.Das, “Ad hoc On-Demand Distance Vector (AODV) Routing”, RFC 3561, 2003.

[7] 守倉 正弘, 久保田 周治, “インプレス標準教科書シリーズ 改訂版 802.11 高速無線 LAN 教科書”, インプレス, 2005

[8] J.Li, C.Blake. D.S.J.D.Couto, H.I.Lee, R.Morris, “Capacity of Ad Hoc Wireless Networks”, Proc. ACM/IEEE MobiCom, July 2001.

[9] V.Jacobson, “Congestion Avoidance and Control”, Proceedings of SIGCOMM ’98, August 1998.

[10] W.Stevens, “TCP Congestion Control”, IETF RFC 2581, 1999.

[11] A.Maor, Y.Mansour, “Ada Vegas: Adaptive Control for TCP Vegas”, IEEE GLOBECOM ‘03, Vol.7 pp.3647-3651, 2003.

[12] L.Ding, X.Wang, Y.Xu, W.Zhang, W.Chen, “Improve Throughput of TCP-Vegas in Multihop Ad Hoc Networks”, Computer Communications, Vo31. pp.2581-2588, 2008.

[13] Z.Fu, X.Meng, S.Lu, “How Bad TCP Can Perform In Mobile Ad Hoc Networks”, ISCC ’02, July 2002.

[14] M.Gerla, K.Tang, R.Bagrodia, “TCP Performance in Wireless Multi-hop Networks”, IEEE WMCSA ’99, February 1999.

[15] Y.Lu, Y.Zhone, B.Bhargava, “Packet Loss in Mobile Ad Hoc Networks”, Technical Report CSD-TR 03-009, Department of Computer Sciences, Purdue University, 2003.

[16] S.Xu, T.Saddawi, “Dose the IEEE 802.11 MAC Protocol Work Well in Multihop Wireless Ad Hoc Networks?”,
[17]K.Chen, Y.Xue, K.Nahrstedt, “On Setting TCP’s Congestion Window Limit in Mobile Ad Hoc Networks”, Proceedings of IEEE ICC, 2003.
[18] K.Nahm, A.Helmy, C.-C.Jay Kuo, “TCP over Multihop 802.11 Networks: Issues and Performance Enhancement”, Proceeding of ACM MobiHoc, May 2005.
[19] H.Zhai, X.Chen, Y.Fang, “Improving Transport Layer Performance in Multihop Ad Hoc Networks by Exploiting MAC Layer Information”, IEEE Transactions on Wireless Communications, Vol. 6, No. 5, May 2007.
[20] H.Zhai, Y.Fang, “Distributed Flow Control and Medium Access in Multihop Ad Hoc Networks”, IEEE Transactions on Mobile Computing, Vol 5, No. 11, November 2006.
[21] S.Xu, T.Saadawi, M.Lee, “Comparison of TCP Reno and Vegas in Wireless Mobile Ad Hoc Networks”, Proceedings of the 25th Annual IEEE Conference on Local Computer Networks, pp. 42-43, 2005.
[22] 銭 飛, “実験で学ぶ QoS ネットワーク技術 NS2 によるネットワークシミュレーション”, 森北出版, 2006.

謝辞

本研究を行うにあたり、日ごろより適切な指導と助言を与えてくださった甲藤二郎教授に心から深く感謝致します。

また、Underwater Sensor Network の研究にあたり、様々なアドバイス、実験環境等を提供してくださった東京海洋大学の近藤逸人准教授に心から感謝致します。

そして、3 年間研究生活を共にした修士 2 年の同期をはじめ、日頃から研究生活をしやすい環境を提供してくださった甲藤研究室の皆さまに心から感謝致します。特に同期のみなさんとは、研究以外でも共にすることが多く、とてもお世話になり、感謝しております。

最後に、私をここまで育てて下さった家族に深く感謝致します。

2010 年 2 月 5 日

飯窪　尚也

発表文献リスト

[1] 飯窪 尚也, 甲藤 二郎, 近藤 逸人, “Underwater Sensor Network のための L3/L4 プロトコルの特性比較”, 電子情報通信学会技術研究報告, vol.107, no. 524, pp. 173-178, 2008 年 3 月.

[2] 飯窪 尚也, 甲藤 二郎, 近藤 逸人, “Underwater Sensor Network における輻輳制御方式の比較検討”, 電子情報通信学会 春季全国大会, B-6-51 , 2008 年 3 月.

[3] Naoya Iikubo, Masayuki Nakatsuka, Jiro Katto, Hayato Kondo, “Improving TCP Performance over Underwater Sensor Networks”, ACM WUWNet ’08, September 2008.

[4] 飯窪 尚也, 甲藤 二郎, “無線マルチホップアドホックネットワークにおける TCP-Vegas の性能改善”, 電子情報通信学会 春季全国大会, B-6-76 , 2009 年 3 月.

[5] 飯窪 尚也, 甲藤 二郎, “無線マルチホップアドホックネットワークにおけるマルチフローに適した TCP-Vegas の特性改善”, 電子情報通信学会 秋季全国大会, B-6-6 , 2009 年 9 月.

[6] 飯窪 尚也, 甲藤 二郎, “無線マルチホップアドホックネットワークにおける TCP-Vegas を拡張した輻輳制御方式”, 電子情報通信学会 春季全国大会, B-6-139 , 2010 年 3 月.

[7] 飯窪 尚也, 甲藤 二郎, “無線マルチホップアドホックネットワークにおける TCP-Vegas を拡張した輻輳制御方式”, 電子情報通信学会技術研究報告, 2010 年 3 月, (投稿中).

[8] 園田 和秀, 飯窪 尚也, 甲藤 二郎, “無線 LAN 環境における 802.11e とウィンドウ制御を併用した TCP-Vegas の特性改善”, 電子情報通信学会 春季全国大会, B-6-144, 2010 年 3 月.

[9] 園田 和秀, 飯窪 尚也, 甲藤 二郎, “無線 LAN 環境における 802.11e とウィンドウ制御を併用した TCP-Vegas の特性改善”, 電子情報通信学会技術研究報告, 2010 年 3 月, (投稿中).

[10] Naoya Iikubo, Jiro Katto, “A Congestion Control Method extending TCP-Vegas over Wireless Ad Hoc Networks”, GLOBECOM ’10, 2010 年 12 月, (投稿中).

[11] 飯窪 尚也, 甲藤 二郎, “無線マルチホップアドホックネットワークにおける TCP-Vegas を拡張した輻輳制御方式”, 電子情報通信学会論文誌, 2011 年 2 月, (投稿中).